Applications of microprocessors in digital high frequency radio communications

Isaac, D. R.

APPLICATIONS OF MICROPROCESSORS IN DIGITAL HIGH FREQUENCY RADIO COMMUNICATIONS

by

D.R. Isaac, B.Sc.

A thesis submitted for the degree of Doctor of Philosophy in the University of Durham, 1981.

Applications of Microprocessors in Digital High Frequency Radio Communications

D R Isaac

Abstract

This thesis describes the application of VLSI devices to channel evaluation and communication techniques over ionospheric radio paths. Digital signal processing techniques using microprocessors and charge coupled devices are described in detail. A novel method for observing interference and fading patterns on HF channels is described. Error control coding schemes and digital modulation techniques are combined in a design for an adaptive modem for use over HF radio links. Results of narrow-band interference measurements, error patterns and coding performance are presented.

Acknowledgements

Firstly, I wish to thank Professor G.G. Roberts for the opportunity of studying in the Applied Physics and Electronics Department in the University of Durham, both as an undergraduate, and as a postgraduate student.

I am particularly grateful to Dr C.T. Spracklen for his help in securing equipment, and for his consistent guidance and advice during the project. Special thanks are due to my girlfriend, Leena, who has provided never-ending encouragement and support throughout the work.

Thanks are also due to Professor E.D.R. Shearman, of the University of Birmingham, for the loan of equipment, and to the Science Research Council for providing a three year studentship.

Glossary of Terms

ACIA	Asynchronous Communications Interface Adapter
A/D	Analogue to Digital
ADC	Analogue to Digital Converter
AM	Amplitude Modulation
ASCII	American Standard Code for Information Interchange
ASK	Amplitude Shift Keying
BCH	Bose Chaudhuri Hocquenghem
BER	Bit Error Rate
BFO	Beat Frequency Oscillator
BS	Block Select
CCD	Charge Coupled Device
CPU	Central Processing Unit
CZT	Chirp-Z Transform
CW	Carrier (or Continuous) Wave
D/A	Digital to Analogue
DAC	Digital to Analogue Converter
DDR	Data Direction Register
DFT	Discrete Fourier Transform
EOF	End of File
EPROM	Erasable Programmable Read Only Memory
FEC	Forward Error Correction
FFT	Fast Fourier Transform
FSK	Frequency Shift Keying
HF	High Frequency
IC	Integrated Circuit
I/O	Input / Output

IRQ	Interrupt Request
LSI	Large Scale Integration
MPU	Microprocessor Unit
MSI	Midwest Scientific Instruments
MUF	Maximum Useable Frequency
NMI	Non Maskable Interrupt
PDR	Peripheral Data Register
PEP	Peak Envelope Power
PIA	Peripheral Interface Adapter
PROM	Programmable Read Only Memory
PSK	Phase Shift Keying
QPSK	Quatenary Phase Shift Keying
RAM	Random Access Memory
RES	Reset
ROM	Read Only Memory
R/W	Read / Write
RX	Receiver
SSB	Single Sideband
SSI	Small Scale Integration
SWTPC	South West Technical Products
TFDM	Time to Frequency Division Multiplexing
TTL	Transistor Transistor Logic
TX	Transmitter
VDU	Visual Display Unit
VHF	Very High Frequency
VLSI	Very Large Scale Integration
VMA	Valid Memory Address

CONTENTS

CHAPTER 1 Introduction

1.1 History

The High Frequency (HF) Radio Band is that part of the electromagnetic spectrum extending from 3 to 30 MHz. Signals transmitted within this frequency range are predominantly propagated via single or multiple reflections from ionized regions within the upper atmosphere. These ionized regions are generally found at heights of 100-400 km. and are collectively known as the ionosphere. The phenomenon of ionospheric propagation has been used for long-distance communication since the pioneering work of Guglielmo Marconi carried out at the beginning of the century. Although the physics of the ionosphere has been studied extensively, the characteristics of the HF channel are often unpredictable. Nevertheless, the ionosphere provides an effective transmission path for a variety of communication traffic and is still widely used to communicate over long distances using a minimum of equipment.

The first successful experiments involving wireless information transmission using electromagnetic waves were based on digital signal representations for reasons of simplicity of the transmitter and receiver structures. This type of communication, called wireless telegraphy, was of great importance during the first half of the 20th century and still exists for certain specialised applications. The invention of electron valves, and later the transistor, gave rise to the development of wireless analogue transmission systems, and, with the discovery of different modulation schemes, the modern

wireless networks for information interchange were evolved; by radio relay, satellite, and ionospheric propagation methods.

The discovery that it was possible to transmit analogue signals by pulse code modulation led to a revival of interest in digital transmission. The successive change from analogue to digital circuits has given rise to the need for digital transmission over wireless networks (1). Considerable efforts are at present being made to investigate the performance of wireless channels in different frequency ranges with respect to digital signal transmission, the HF band being no exception.

Because of the problems associated with digital transmission over HF radio links, this medium has not been used successfully for reliable communication of medium- and high-speed data traffic. Certain investigations and experiments were carried out, mainly in the United States, during the 1950's and early 1960's (5,6), but were largely abandoned in favour of satellite communication links which yield higher bit rates with a reduced probability of error. However, many users, faced with problems of the cost and vulnerability of satellites, are prepared to tolerate a reduction in bit rate in exchange for economical systems using a virtually indestructive communications medium. Indeed, for mobiles operating over long-distance circuits, such as ships and aircraft, the HF path is almost the only alternative to satellite communication.

The recent advances in digital integrated circuit technology have enabled substantial reductions to be made in both the size

and the cost of communication systems (2) and it seems only logical that HF radio systems should also benefit from such developments. The work described in this thesis is concerned with the applications of VLSI (Very Large Scale Integration) technology to channel evaluation and data communication using the HF radio path. It is shown that systems which previously required large amounts of expensive analogue equipment can be realised at lower cost and with increased flexibility using nearly all digital techniques.

1.2 The Ionosphere

An effect of solar radiation is to cause ionisation of certain regions of the earth's atmosphere. This ionisation of gas molecules results in an electron density profile which is non-uniform with height. The region of the atmosphere having the highest electron density is known as the 'ionosphere', and it is this region which is responsible for the phenomenon of ionospheric propagation.

When a radio wave transmitted from the surface of the earth encounters a region of intense ionisation, it will be diverted from its original path by a refractive mechanism. The wave will also be considerably attenuated. At certain frequencies the refractive process is sufficient to 'bend' the wave through an angle exceeding 180^{o}, at which point the attenuated wave will be returned to earth. The frequencies at which this phenomenon most commonly occurs lie within the range 3-30 MHz (the HF radio band). Because the refractive process appears to be one of reflection, this phenomemon is often referred to as 'ionospheric

reflection', a term which is subsequently used in the text.

The vertical electron density profile is non-linear; there are regions of ionisation which are more intense than others. The heights of these regions depend on many factors, one of which is the intensity of the solar radiation which varies from day to night, and with the seasons. In the long term, they are dependent on variations of the solar (sunspot) cycle. There are, however, three regions of chief importance, referred to as 'layers'. These are the E-layer at 120 km. (all figures are approximate), the F1-layer at 200 km., and the F2-layer at 300-400 km. At night, and in mid-winter, the F1 and F2 layers combine to form a single F-layer at 250 km. Below the E-layer there is a D-layer (at 50-90 km.) which is generally more important as an absorber than a reflector of radio waves since the attenuation at this altitude is somewhat greater than at higher regions.

Figure 1.1 is a simplified representation of a typical HF propagation path showing the ionosphere comprising two ionized layers, the E-layer and the F-layer. The signal is returned to earth via a single-hop E-layer mode together with single- and double-hop F-layer modes. Consequently, a short transmitted pulse will have three components when detected at the receiver, each component itself exhibiting time dispersion which causes a corresponding broadening of each received pulse. This latter form of dispersion is due to the fact that the transmitted energy illuminates the ionospheric layers over a relatively large area rather than at a single point; thus the energy of a given

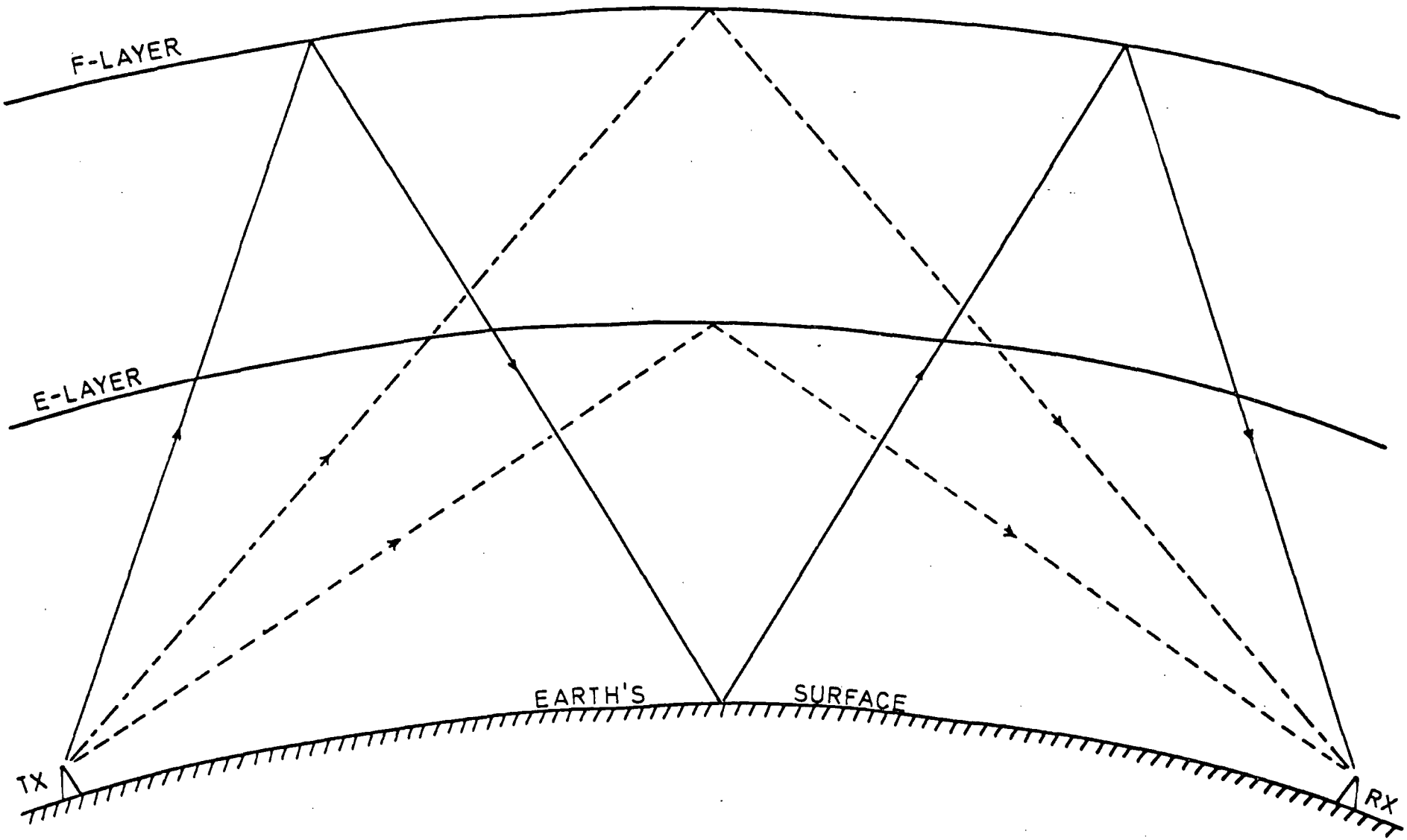


Figure 1.1 Simplified HF Propagation Path

received component can be considered as being made up from the energy of a large number of elemental returns, each with a slightly different propagation time, integrated over an appropriate area of the ionospheric layer.

There are many problems concerning the communicator who wishes to use the ionosphere as the transmission medium. Attempts have been made to model the HF channel (3), but the (largely unpredictable) time variation of the channel characteristics considerably increase the difficulties. Some of the problems are now discussed, and suggestions are included for some ways of combating the detrimental effects of the channel.

1.3 Problems of the HF channel

The problems associated with digital communication via ionospheric propagation may be classified into two broad categories:

(1) Multiple reflections from different layers in the ionosphere give rise to differential time delays in the received signal which results in intersymbol interference and fading.

(2) Noise, both natural and man-made, causes errors in the transmitted data stream.

Both problems are of equal concern in that they corrupt the received data to a sometimes considerable extent. Each will now be discussed separately.

1.4 Multipath propagation

The problem of intersymbol interference is inherent in any digital communications link but is especially important over HF radio links due to the comparatively long relative time delays introduced by multipath (or multimode) propagation. For a typical HF link of, say, 1000 km., the time delay between the shortest and the longest propagation path may be in the order of 2 to 3 milliseconds, the effect of which is to cause intersymbol interference of such severity as to considerably reduce the maximum allowable transmission rate. The time variations of the amplitudes and phases of the individual modes will be independent and any one mode may be dominant at a particular instant. In addition, phase cancellations may cause fading of the received signal. The result of multipath propagation, in any event, is to introduce errors in the transmitted data. Whether or not it is possible to recover these errors is dependent on the interpretation of the data at the receiver.

1.4.1 Overcoming multipath problems

Broadly speaking, the detrimental effects of multipath propagation may be reduced in four ways:

(1) Choice of a suitable operating frequency

(2) Time-to-frequency division multiplexing

(3) Channel equalization

(4) Diversity techniques

The highest frequency at which ionospheric propagation is possible for a given range is termed the Maximum Useable

Frequency (or MUF) and varies with the time of day and with the seasons as well as being severely affected in an unpredictable manner by solar flares etc. For an HF radio link, as the frequency of operation is increased, the number of propagation modes is reduced until single-mode propagation results at a frequency just below the MUF. It is possible to predict the MUF on a long-term basis, but the short-term fluctuations may considerably alter the prediction figure (4). However, if a large number of channels is available to the user, considerable improvements can result from the correct choice of operating frequency.

Time-to-frequency division multipexing (TFDM) is a technique used to increase the transmission rate while keeping the signal element duration (frame rate) constant (5,6). The data stream is divided into short blocks for transmission over a number of frequency-parallel sub-channels, all contained within the allocated voice channel, and orthogonally spaced to avoid co-channel inteference. Because the frame rate is constant, the signal element duration can be greater than the multipath spread, and the data rate may be increased by a factor which is the number of sub-channels that can be fitted in to the available channel capacity.

Channel equalization involves the construction of a matched filter to combat the degrading effects of the channel. Such a filter normally consists of a tapped delay line, whose tap outputs are weighted and summed to provide the filter output. Convolution of the received signal with the impulse response of

the filter should result in the original signal. Because of the time-varying nature of the HF medium, such an equaliser must be adaptive in nature, the number and position of the delay-line taps being continuously updated to allow for changes in the transmission medium (7). In practice, adaptive equalisers are difficult to implement and can be extremely costly.

Diversity may be employed in time, frequency, or space. A simple time diversity technique is simply to send the message more than once, thereby increasing the probability of correct reception. Dual frequency diversity requires transmission over two channels simultaneously. In one method, both channels are monitored and the channel which yields the most favourable error performance is selected. Two channel filters are therefore required at the receiver and the transmitted spectrum is doubled in width resulting in inefficient spectrum utilisation. Space diversity is the technique most commonly employed (8) and is based on the (usually correct) assumption that the time-variation of the phase cancellations at one point in space will be different to those occurring at another point which is located at a distance away which is comparable to the transmission wavelength. This method is quite effective but requires two receivers and two antennas. In all diversity techniques the problem of designing efficient combiners is a considerable one, and in any case a doubling of the resources is required in some sense.

1.5 Noise

The other major source of errors is noise, which may be naturally occurring, or which may arise from some sort of man-made disturbance. This latter type is often a result of congestion within the HF spectrum creating narrow-band interference from other users of the HF medium. A channel may be allocated to several users, who may interfere with each other if there is propagation between them. Serial data transmission systems (eg. radio telegraphy) suffer considerably from this type of interference (9). Broad band noise may be caused by machinery operating in the vicinity of the receiver although this sort of disturbance is normally localised in nature. A more common source of broad-band noise is that resulting from electric storms or other ionospheric disturbances which may generate a considerable amount of radiated energy in the HF spectrum. This type of noise generally occurs in bursts and is especially prevelant during the summer.

1.5.1 Overcoming noise problems

The detrimental effects of noise on the channel may be dealt with in a number of ways. Front end linearity in the analogue portion of the receiver is an important consideration when attempting to receive signals in the presence of heavy interference. Space diversity may assist in combating localised interference while time and frequency diversity may be effective in avoiding broad- and narrow-band noise respectively. A technique closely associated with TFDM is known as frequency agility which has been found to be effective in combating the effects of narrow-band noise (10) whereby a sounding technique is

used to select the "quietest" channel or in-band subchannel for subsequent data transmission. Sounding techniques are also used to select the best channel from a set of allocated channels by transmitting a sounding signal on each of the channels (11). The receiver then assesses the suitability of each of the channels for data communications and advises the transmitter via a feedback link.

Diversity methods imply some redundancy in their respective domains. A binary system may employ time diversity by the addition of redundant bits to the transmitted data. The redundancy may then be used as an aid in reducing the detrimental effects of noise on the transmitted bit stream.

A trade-off must first be made between the degree of fidelity required in the received data and the resulting reduction in the transmission rate caused by the addition of redundancy. Once this has been determined, the redundant bits can be added in a systematic manner which will then enable certain statistical categories of errors to be corrected and/or detected (12). As an example, one method is to use a block coding scheme in which the data stream is divided into a number of blocks each of which is k bits in length, and each of which is mapped into a 'codeword' of n bits (n>k) by adding a number **n-k** bits which are the results of n-k modulo-2 additions (or parity checks) on the original k data bits. The resulting codewords then form part of a linear block code which obeys a fixed set of parity-check rules. If the parity checks do not agree on reception, errors have occurred and the appropriate bits may be corrected. Some of

the most powerful block codes known are of length $n = 2^{m-1}$ (m is an integer) and are capable of correcting up to t random errors occurring within a code word and require, at most, the addition of **mt** check bits (13). Errors occuring on HF radio links are generally greater than one bit in length and it is therefore preferable to use codes which can correct bursts of errors. However, random-error-correcting codes can be used very effectively to correct bursts of errors by interleaving groups of codewords such that a burst of consecutive errors is distributed over several codewords, the errors appearing as random to the decoder.

In general, then, the ionosphere can be considered to be an anisotropic time-varying medium which causes the HF path to exhibit error rates far greater than those found in other communication systems (14). The results of multipath propagation and noise are to introduce large numbers of random and burst errors which may sometimes be so great as to render the channel virtually useless. Indeed, to an outsider, communicating via such a medium may seem a near impossible task. However, it seems that adaptive schemes combining several of the above-mentioned techniques may yield substantial performance improvements. Until recently, adaptive schemes have been complex and costly to implement, and almost all systems described in the literature have not had adaptive capability (15).

1.6 Microprocessors and HF Radio Systems

One of the most significant products of the VLSI technology previously discussed is the microprocessor. This is a monolithic device which is obtainable at low cost and which may be made to perform a wide variety of processing tasks by the choice of a suitable sequence of instructions stored in a read-only memory (ROM) unit. The system is configured such that the microprocessor may access the stored instructions and execute them accordingly. Major system changes may be implemented by simply modifying the instruction sequence (the 'software'). A microprocessor system may be made adaptive by determining that the order of execution of the instruction sequence is dependent on previous and/or present events.

This thesis describes the applications of microprocessor techniques to several aspects of digital communications over HF radio channels. The next chapter discusses the implementation of several discrete-time signal processing techniques which are used in subsequent chapters in this thesis. An HF "spectrogram" sounding technique is described in chapter 3, from which a visual display of the time-frequency characteristics of an HF voice channel may be obtained. Information from the display may subsequently be used to determine the suitability of the channel for data transmission.

During the course of the project, it became evident that the data processing requirements for some applications exceeded the capability of a single microprocessor unit. It was for this reason that a multiprocessor system (chapter 4) was developed, in

which several 'slave' processor units operate under control of a central, or 'master' processor. This configuration allows a considerable increase in data throughput over a single processor system. Various error control coding schemes were investigated (chapter 5), leading to the implementation and subsequent field testing of several software-based schemes for the correction of errors occurring over HF radio data links. Chapter 6 presents some results obtained from a detailed investigation into interference phenomena observed in an HF communications channel; the implications for the design of data modems are discussed.

The ideas and results obtained from previous chapters in this thesis were combined in chapter 7 to implement an adaptive HF data modem for data transmission over HF radio links. Signal processing, error-control coding and multiprocessor techniques are used in a low-cost system for medium-speed communication which attempts to overcome the detrimental effects of the HF channel. Some results obtained using the equipment are described in chapter 8.

1.7 Conclusion

The last decade has seen a resurgence of interest in communication via the ionospheric medium. The problems associated with data transmission via this unpredictable channel are considerable, and many techniques have been developed in attempts to combat the degradation imposed on the propagated signal. Some of these techniques have been utilised in systems which have been designed exclusively for HF radio data transmission, but have been complex and costly to implement.

Microprocessors have made substantial inroads in several fields of digital data communications and are well suited to systems requiring adaptability. This thesis describes some applications of these devices to data transmission over HF radio links and shows that considerable savings in hardware requirements may be made by using a primarily software-based approach to system design.

CHAPTER 2 Signal Processing Techniques

2.1 Introduction

The emergence of digital signal processing as a major discipline began in the mid-1960's when high speed digital computers became available for research and development work (16). Many concepts that form the theoretical basis of digital signal processing, such as the z-transform and Fourier analysis, had been familiar, however, to engineers for a long time. In the ensuing years, the field has matured considerably and its development is intimately tied with technological advancement in device design and fabrication.

Many of the signal processing requirements of HF radio communications systems may be realised using digital techniques implemented with the aid of microprocessors. In general, an analogue signal is sampled using an analogue-to-digital converter, the resulting set of samples is processed by a microprocessor-based system, then converted back to analogue form by a digital-to-analogue converter. The efficiency and accuracy of the signal processing depends to a great extent on the design of the system software.

This chapter introduces the microprocessor chosen for the subsequent work described in this thesis. The design and operation of a novel system used for data storage on magnetic tapes is then discussed. More complex techniques are then introduced, based on the Discrete Fourier Transform (DFT). The DFT plays an important role in the analysis, the design, and the

implementation of digital signal processing systems concerned with HF radio systems and is used in several of the applications described in following chapters. Software and hardware implementations of efficient algorithms for the computation of the DFT are investigated and compared. An assessment is made of the suitability of such implementations for demodulation and spectral analysis in HF radio systems.

2.2 The Microprocessor

The choice of a suitable microprocessor was a primary consideration in the development of the project. A device was required having a comprehensive instruction set, fast execution speed, and for which support facilities were available. The Motorola M6800 (17) satisfied these requirements and was chosen in view of the following merits:

(1) The M6800 has a powerful and versatile instruction set (72 basic instructions and 7 addressing modes).

(2) Several economical and flexible development systems were available.

(3) A cross-assembler and simulator were available on the university computer.

(4) Software packages, ancillary components, and documentation were readily available.

(5) Execution time is fast (nominal clock frequency = 1MHz; average instruction time = 4 cycles (approx.)).

The M6800 is a monolithic microprocessor having an 8-bit data bus and a 16-bit address bus. It has six internal registers: the A accumulator (8 bits), the B accumulator (8

bits), an index register (16 bits), a program counter (8 bits), a stack pointer (16 bits) and a condition code (status) register (8 bits). The device requires a non-overlapping bi-phase clock for normal operation.

Two systems based on this device were constructed for the development of software during the project (18). Each system consisted of (1) a "motherboard", with 7 card slots available on the main bus (called the SS-50), and with 8 slots available on the I/O bus (called the SS-30), (2) a CPU card containing the microprocessor, crystal oscillator, baud rate generator, ROM (containing a monitor program), and 128 bytes of "scratchpad" RAM, (3) two 16 kbyte RAM cards, and (4) a serial interface card enabling the system to be controlled from a teletype or a VDU. An EPROM programming card was available, allowing machine code programs (object data) to be transferred from RAM to 2 kbyte EPROMs under software control.

The only mass storage system available during the first few months of the project was a teletype paper tape punch/reader having an operating speed of 110 baud. An editor and an assembler were available as aids to program development; each of these packages, however, required approximately half an hour to load into memory using the paper tape reader. It became evident that a more efficient means of mass storage was necessary if reasonable progress was to be made. It was for this reason that a magnetic tape storage system was subsequently adopted. A novel interface to the tape unit was developed which required a minimum of additional hardware (only 2 I/O lines of a Peripheral

Interface Adapter (PIA) IC were needed). The design of this interface is discussed in the following section and is also described in the publication to be found in Appendix 4 (reference 19).

2.3 Cassette interface

The "Kansas City" standard is commonly regarded as an internationally accepted standard for data recording on audio cassettes. The baseband data signal is used to modulate carriers of 1400 Hz and 2800 Hz using FSK at a rate of 350 bps., ie a logic '0' is represented as 8 cycles of the higher frequency and a logic '1' as 4 cycles of the lower frequency. It was initially decided to design an interface based on this standard but this was later developed into a higher speed system.

The PIA (Peripheral Interface Adapter (20)) is a device which provides an interface between the 6800 microprocessor bus and two external data ports of 8 lines each (the 'A' and the 'B' ports). Each of the 16 lines is compatible with standard TTL logic families and may be programmed for input or output by setting the state of the corresponding bit in an internal "data direction register". For the cassette interface, line A_7 was programmed for output and line B_7 for input. Assembly level software was written which would take data from a specified region of memory and generate the appropriate FSK signal at the output line A_7. Cycle timing of the rectangular modulated signal was achieved using software timing loops and data was transmitted in byte format using a '0' start bit and a '1' stop bit. The TTL level FSK output was connected directly to the

microphone input of the cassette recorder, which incorporated an automatic record level preamplifier. On playback, the audio output from the extension speaker socket was connected directly to the input line B_7 which was continuously sampled by the processor. The period of a waveform cycle was determined by comparing the number of '1' samples counted with a mean threshold value. If the number of samples counted exceeded this value the cycle was of 1400 Hz; if less it was 2800 Hz. The sampling rate was approximately 12 kHz. A decision as to the state of a demodulated data bit could be made after receiving the appropriate number of cycles. The average error rate from this system on playback was found to be better than 1.5×10^{-6}. (ie. less than 1 error per 40 minute recording)

Some experimentation was carried out with adjustment of the recording frequencies and number of cycles per bit. A reliable and much faster system was found to result from using only single cycles of the original two frequencies; ie. a logic '0' takes only half the time of a logic '1' to load and store. An example of the signal generated for the data sequence 10110 is shown in figure 2.1. If the data contains equal numbers of 1's and 0's, the mean bit rate is 1600 bps, more than 5 times that of the Kansas City standard.

It was decided to incorporate error checking by recording the data according to the Motorola "MIKBUG" standard format for recording object data on punched paper tape (21). This format requires data to be transmitted in ASCII character records arranged as in figure 2.2. All information in the record is

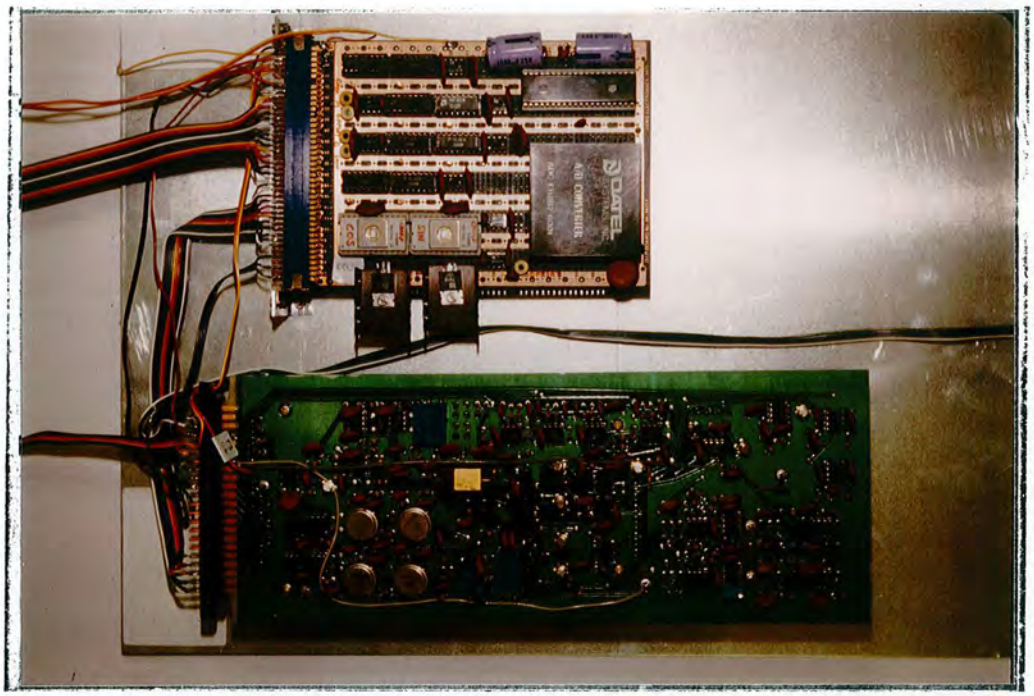


PHOTO 2.1. SHOWING POST-MULTIPLIER UNIT INTERFACED TO CCD EVALUATION MODULE.

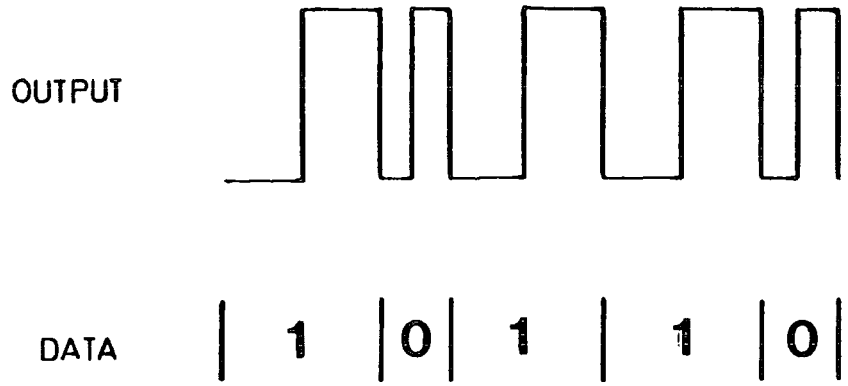


Figure 2.1 Cassette Data Format

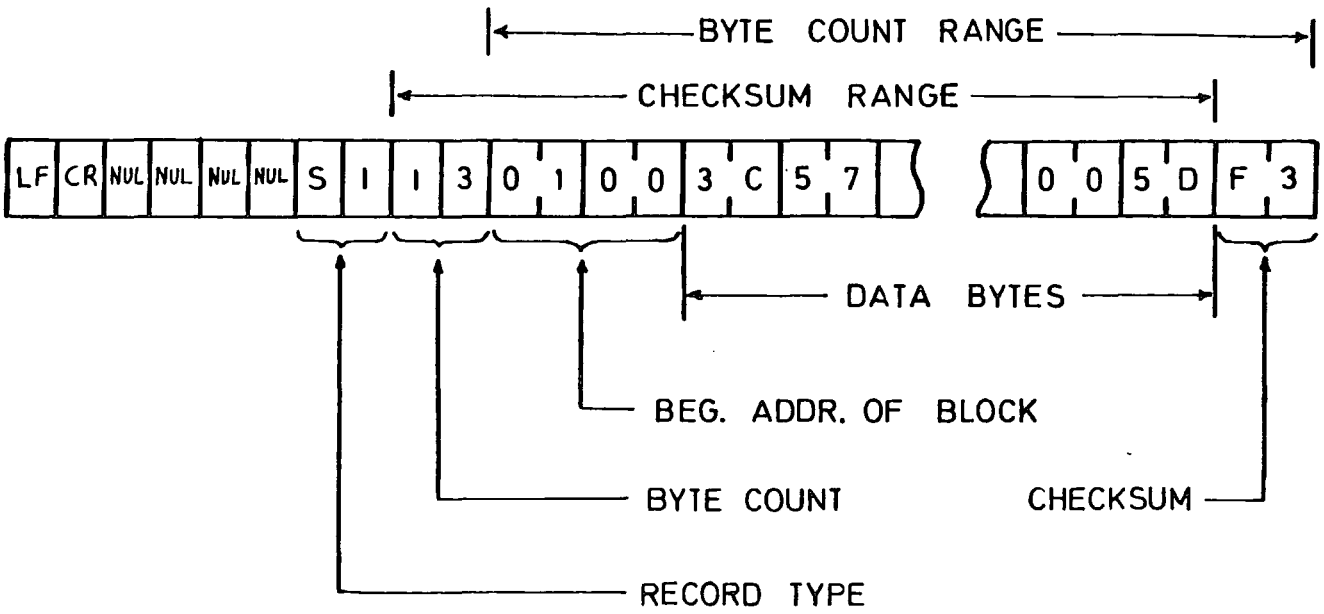


Figure 2.2. "MIKBUG" Tape Format

represented as hexadecimal data. The beginning of the record contains the record type (S1), a byte count which covers all the bytes that follow, and the start address of the data block. Data bytes follow which represent the object data to be stored in memory beginning at the block address and stored in sequential memory locations that follow. At the end of the record is a checksum, which is the 1's complement of the summation (mod 256) of all data bytes in the record, plus the byte count and block address. This value is checked as data is loaded and a '?' is printed if an error is encountered. The tape can then be rewound a few blocks and the faulty block reloaded. The end-of-file (EOF) terminates the data and consists of the characters "S9". The EOF terminates a tape load function.

An assembly listing of the tape store and load programs is included in the published article at the end of this thesis (Appendix 3). The high-speed interface showed error rates at least as good as the Kansas City version, and was used successfully for several months before the purchase of a flexible ('floppy') disc drive and interface board.

During the second year of the project, a triple disc drive and interface board were obtained to facilitate faster file access. Each flexible disc could contain 80 kbytes of storage; it was therefore possible to have access to 240 kbytes of data at any one time. Commercially available software was obtained, including a co-resident assembler, editor, and BASIC interpreter, to assist program development. The teletype was replaced with a VDU for communication with the development system and hard copy

output was obtained from a thermal printer attached to an additional serial interface card connected to the SS-30 bus.

2.4 The Discrete Fourier Transform

It has already been mentioned that the DFT plays an important role in the analysis and processing of HF radio signals. The DFT of a finite length sequence, $\{x(n)\}$, is defined as

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn} \quad , \quad k = 0,1, \ldots , N-1 \quad (2.1)$$

where $W_N = \exp(-j2\pi/N)$.

The resulting sequence, $\{X(k)\}$, is the Fourier representation of the original input sequence. The following sections of this chapter discuss efficient microprocessor implementations of the DFT using software alone, using hardware arithmetic, and with a Charge Coupled Device (CCD), each of which was used in a different application. A description of the algorithms is given, followed by a discussion of the implementations, and finally a comparison of the implementation efficiencies.

2.5 FFT Algorithm

The direct computation of the DFT requires $4N^2$ real multiplications and $N(4N-2)$ real additions, or, equivalently, N^2 complex multiplications and $N(N-1)$ complex additions (22-24). The number of multiplications required is generally accepted as being a meaningful measure of the complexity, or, of the time required to implement a computational algorithm. The amount of computation required to evaluate the DFT directly is approximately

proportional to N^2; the computation time required to compute the DFT by this method therefore becomes very large for large values of N.

It is possible to reduce the number of computations required to evaluate the DFT by decomposing the sequence {x(n)} into successively smaller subsequences. Algorithms based on this principle are called "decimation in time" algorithms (22,23). If N is an even number, we can consider computing X(k) by separating {x(n)} into two N/2-point sequences consisting of the even- and odd-numbered points in x(n).

We then obtain

$$X(k) = \sum_{n \text{ even}} x(n)W_N^{kn} + \sum_{n \text{ odd}} x(n)W_N^{kn}$$

Substituting n=2r for n even and n=2r+1 for n odd gives

$$X(k) = \sum_{r=0}^{(N/2)-1} x(2r)W_N^{2rk} + \sum_{r=0}^{(N/2)-1} x(2r+1)W_N^{(2r+1)k}$$

$$= \sum_{r=0}^{(N/2)-1} x(2r)(W_N^2)^{rk} + W_N^k \sum_{r=0}^{(N/2)-1} x(2r+1)(W_N^2)^{rk}$$

but $W_N^2 = \exp(-2j(2\pi/N)) = \exp(-j2\pi/(N/2)) = W_{N/2}$

$$\therefore X(k) = \sum_{r=0}^{(N/2)-1} x(2r)W_{N/2}^{rk} + W_N^k \sum_{r=0}^{(N/2)-1} x(2r+1)W_{N/2}^{rk} \qquad (2.2)$$

$$= G(k) + W_N^k H(k)$$

Each of the sums in equation 2.2 is recognised as an N/2 point DFT. If N is an integer power of 2, each of the sums may be decomposed further into two (N/4) point DFTs and so on until the

stage is reached where there are N/2 2-point DFTs (or "butterfly" computations) to be performed. There will be $\log_2 N$ stages of decomposition in all. The basic butterfly computation is illustrated in figure 2.3 and is described by the following equations:

$$X_{m+1}(p) = X_m(p) + W_N^r X_m(q)$$
$$X_{m+1}(q) = X_m(p) - W_N^r X_m(q)$$

Only one complex multiplication is involved in mapping the point $X_m(p)$ and $X_m(q)$ to $X_{m+1}(p)$ and $X_{m+1}(q)$ respectively. Figure 2.4 shows a graphical representation of the evaluation of an 8-point DFT using the butterfly computation of figure 2.3.

The total number of multiplications required for the above algorithm is $(N/2)\log_2 N$ compared with N^2 multiplications required for direct evaluation of the DFT. The algorithm was derived by Cooley and Tukey in 1965 (25), and is often referred to as the Fast Fourier Transform algorithm (FFT). A flow diagram for the FFT decimation in time algorithm is shown in figure 2.5. Three variables, i, m, and and l, are required to index through the complex array A(*). There are $r=\log_2 N$ stages of computation and N/2 butterfly computations to be performed at each stage.

2.5.1 Bit-reversal shuffling

If the above algorithm is to be used to produce an output sequence in sequential order, it is necessary to store the input data in non-sequential order. In determining the position of x(n) in the input array, the bits of the binary representation of the index n must be reversed. For example, for a 16 point transform,

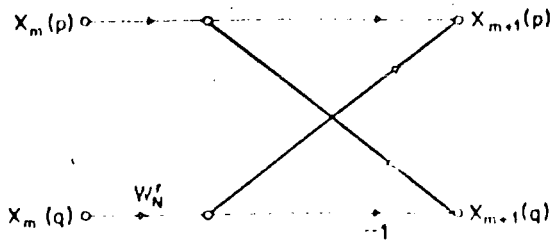


Figure 2.3. Basic Butterfly Computation

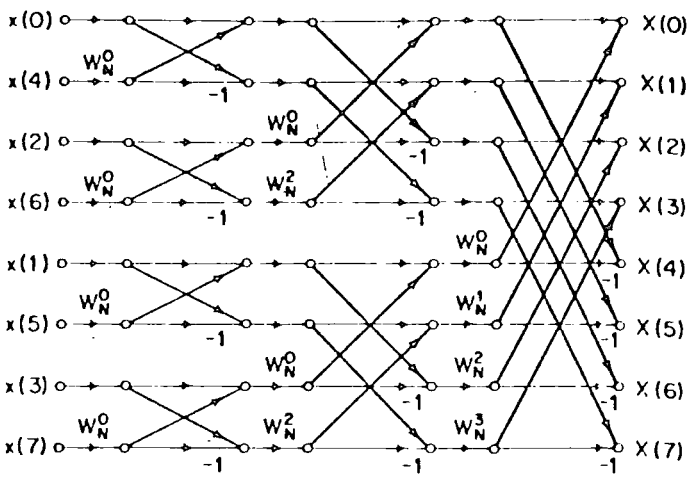
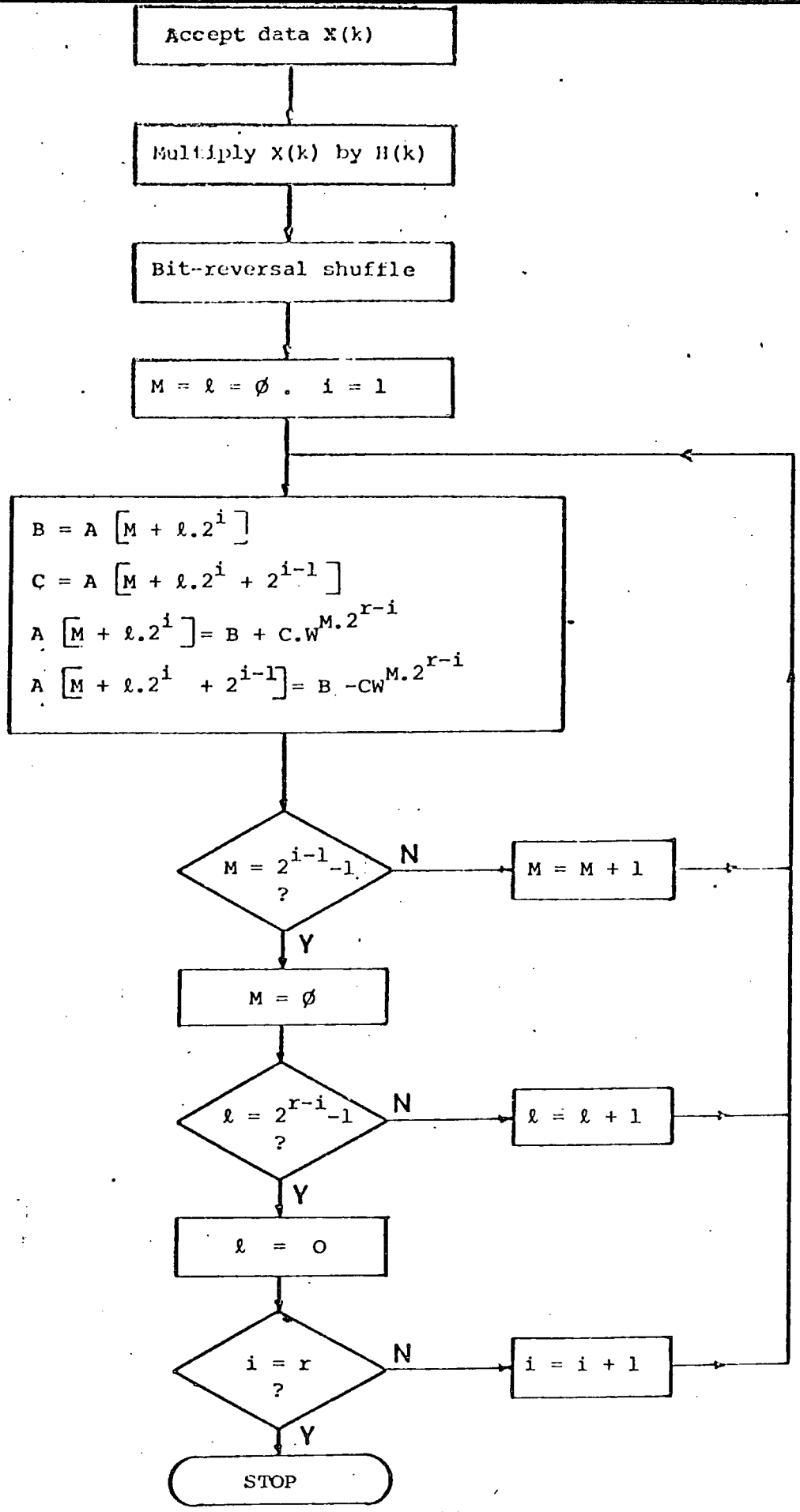


Figure 2.4. Flowgraph of 8-point DFT using butterfly computation of figure 2.3 .



All multiplications are complex. $W^k = e^{-j\left(\frac{2\pi}{N}\right)k}$

FIGURE 2.5. FFT ALGORITHM FLOWCHART

four bits are required for each index and an index $(b_3 b_2 b_1 b_0)$ becomes $(b_0 b_1 b_2 b_3)$. The necessity for bit-reversal shuffling of the sequence x(n) is a result of the manner in which the DFT computation is decomposed into successively smaller DFT computations. Figure 2.6 shows the flowchart of an algorithm which will shuffle the contents of the array x(i) , i=0,1,2 ... N-1 in bit-reversed order; the result will be contained in the same array. Each of the numbers x(i) may be complex. Three integer variables, i, j, and k are used to index through the data and the complex variable X is used as a temporary storage location. This algorithm was implemented in BASIC and assembly level language for use with the respective FFT programs.

2.6 CZT Algorithm

Another algorithm for evaluating the DFT, called the "chirp-z" transform (CZT), was derived in 1969 by Rabiner and Schafer (26), and is not restricted to integral powers of N as is the FFT. The expression for the DFT in 2.1 is:

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j2\pi nk/N} \quad , \quad k = 0,1, \ldots ,N-1$$

where both x(n) and X(n) may be complex. Using Bluestein's identity (27)

$$nk = \frac{1}{2}\left[n^2 + k^2 - (k-n)^2\right]$$

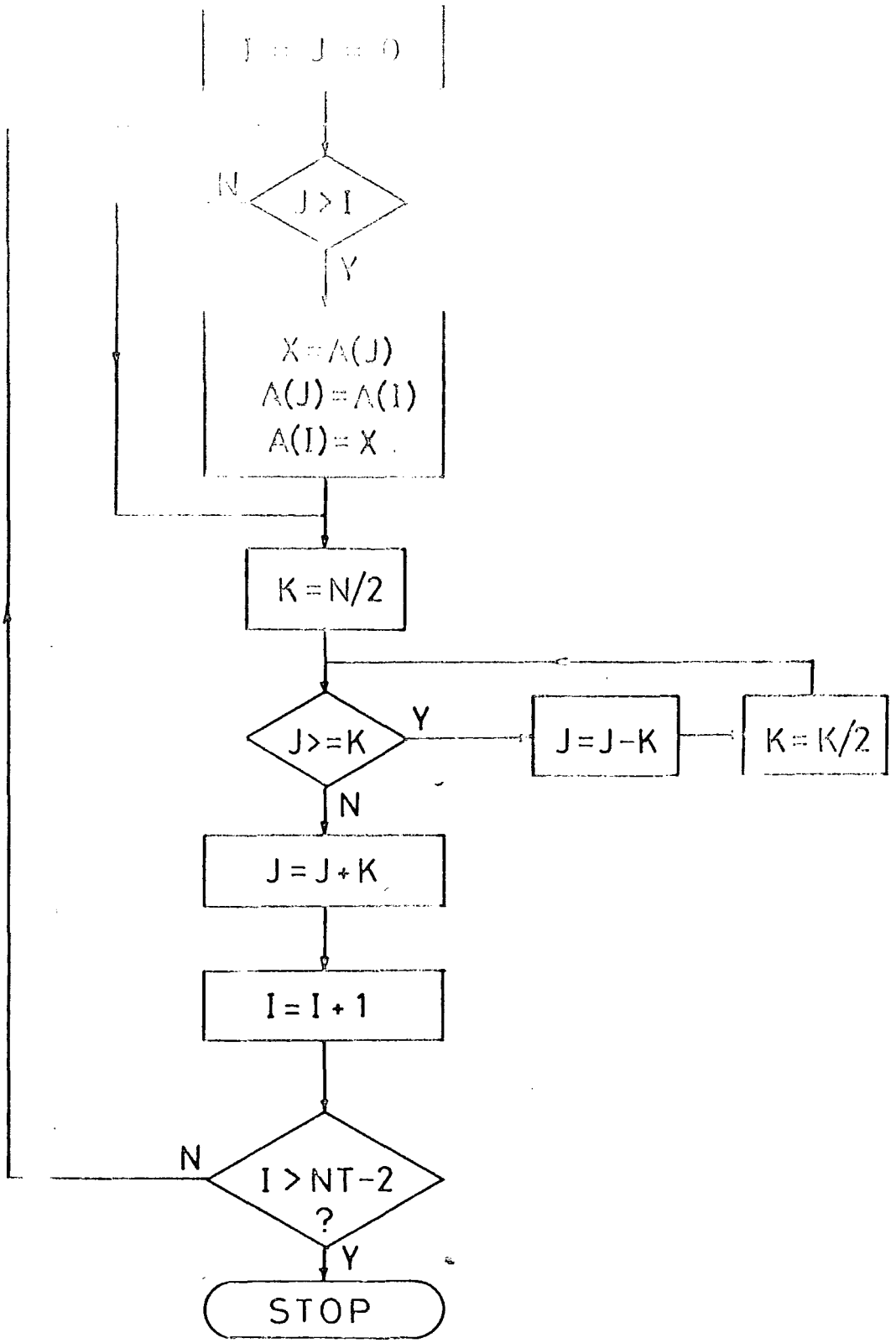


FIGURE 2.6. BIT-REVERSAL SHUFFLING

we obtain

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-j\pi n^2/N} \; e^{j\pi(k-n)^2/N} \; e^{-j\pi k^2/N}$$

$$= e^{-j\pi k^2/N} \sum_{n=0}^{N-1} g(n) \; e^{j\pi(k-n)^2/N} \quad , \; k=0,1, \ldots \; N-1 \qquad (2.3)$$

Equation (2.3) is the expression for the CZT. Three stages of computation are required

(i) Multiply each term x(n) by the complex factor $\exp(-j\pi n^2/N)$ to produce a new sequence g(n).

(ii) Perform a discrete convolution between the sequence g(n) and the sequence $\exp(j\pi n^2/N)$.

(ii) Multiply the resulting output sequence by the factor $\exp(-j\pi k^2/N)$ for each point of X(k).

The sequences $\exp(-j\pi n^2/N)$ and $\exp(-j\pi k^2/N)$ can be thought of as complex exponential sequences with linearly increasing frequency. Such signals are called "chirp" signals; hence the name "chirp-z" transform. A method for implementing the CZT using a charge-coupled device is described later in this chapter.

2.7 Implementations

Four methods for implementing the DFT using the two algorithms described were investigated and used in various applications described elsewhere in this thesis:

(i) Implementation of FFT algorithm using a co-resident BASIC interpreter

(ii) Assembly-level implementation of the FFT using software

arithmetic

(iii) Assembly-level implementation of the FFT using hardware multiply/divide unit

(iv) Hardware implementation of the CZT using a charge coupled device

2.8 FFT Implemenation in BASIC

A program to evaluate the DFT of an N-point real sequence is shown in the listing in appendix 2. The following results may be obtained from this program: printout of input data, printout of output data (real & imaginary components), plot of output data (real & imaginary), printout of power spectrum and plot of power spectrum. A window function may be applied to the input data if a power spectrum is required. A special feature of this program is that the DFT of the original N-point input sequence is evaluated by performing a single N/2-point FFT computation. The spectrum of a purely real sequence exhibits complex conjugate symmetry and it is therefore only worthwhile computing the positive frequency half of the spectrum.

Any asymmetric function may be formed as the sum of two functions, symmetric about some suitable axis, one posessing even and the other odd symmetry. Using this fact it is possible to simultaneously compute the spectra of two N/2-point real sequences using only one N/2-point complex FFT. One sequence y(n) is entered as the real components in an N/2-point complex array; the other, z(n), is entered as the imaginary components. The transform, X(k), of the sum x(n)=y(n)+jz(n) is asymmetric but it is possible to use the symmetry property to obtain the spectra

Y(k) and Z(k) of y(n) and z(n) respectively by a manipulation of the sequence X(k). If the sequences y(n) and z(n) are the even and odd numbered points respectively of an N-point real input sequence, it is possible to obtain the spectrum of this sequence from Y(k) and Z(k). This method is described in reference (24).

The even-numbered points of the real input sequence are entered as the real components of the complex array A(2,N/2), and the odd-numbered points are entered as the imaginary components. An N/2-point DFT is performed by the subroutine at statement 1500 which shuffles the data in bit-reversed order, then evaluates an N/2 point FFT. The subroutine at statement 520 manipulates the transformed data to obtain the spectra of the two real sequences. The positive half of the spectrum of the original input data sequence is then found using the subroutine beginning at statement 130. Real and imaginary components of the spectrum may be listed and/or plotted if required. The power spectrum of the transform is found by evaluating the square root of the sum of the squares of the real and imaginary components of each point in the output sequence.

2.9 FFT Implementation in assembler

The FFT algorithm of figure 2.5 was implemented in M6800 assembly language in order to obtain an increase in computation speed over a high level language implementation. A listing of the program is given in appendix 2 and is capable of evaluating the DFT of N complex data points where N is an integer power of 2 between 8 and 256 inclusive. The real and imaginary components of each point are quantised to 8 bits each but arithmetic operations

are carried out to 16 bit accuracy to eliminate quantisation errors introduced between stages of the decimation-in-time implementation. 1 kbyte of memory was therefore necessary to accomodate 256 complex points as 2 bytes were required for each of the real and imaginary components. Data is arranged in the table as follows:

address	data
DBASE+4n	f(n) (real) MSB
DBASE+4*n+1	f(n) (real) LSB
DBASE+4*n+2	f(n) (imag) MSB
DBASE+4*n+3	f(n) (imag) LSB

To avoid computing the sine and cosine values for the weighting factors, a table of lookups was used. Values in the table corresponded to the functions:

$$W_{RE}(n) = \cos(2\pi n/256) \quad n = 0,1,2, \ldots 127$$
$$W_{IM}(n) = -\sin(2\pi n/256) \quad "$$

Each complex point was stored as two 16 bit 2s complement numbers in a similar format to the data storage format described above. 512 bytes were therefore required for lookup table storage. The binary representations of the weighting factors were as follows:

W	binary
+1	0100000000000000
0	0000000000000000
-1	1100000000000000

The program is invoked by first loading the data into the data table, loading r ($=\log_2 N$) with a value between 3 and 8 and executing the call "JSR FFT". The data is first shuffled in

bit-reversed order by the subroutine SHUF which is a direct implementation of the flowchart of figure 2.6. Successive butterfly computations are then performed in-place each of which involves 4 real 16 x 16 bit multiplications, 3 real 16 + 16 bit additions and 3 real 16 - 16 bit subtractions.

Two implementations were evaluated; the first used a software multiplication routine based on the Booth algorithm of reference (28); the second used a hardware multiply/ divide unit, described in section 2.11. Execution times for both implementations are tabulated in section 2.12. A further improvement in computational efficiency can be obtained by observing that the weighting factor W^0 has real and imaginary coefficients of 1 and 0 respectively and no multiplications are required for this case. The total number of real multiplications required for the N-point DFT therefore reduces to

$$4 \times ((N/2)\log_2 N - (N-1))$$

2.10 Implementation of the CZT

As discussed in section 2.6, the CZT algorithm involves three stages of computation: pre-multiplication, convolution, and post-multiplication (29). The block diagram of a complete transform based on the CZT algorithm is shown in figure 2.7. Pre-multiplication is accomplished by the multipliers to the left of the diagram and post-multiplication by those on the right. The major computing task is the convolution portion; the Reticon R5601 quad chirped transversal filter (30) has been designed to perform this task. This device contains two separate 512-stage MOS charge coupled devices which are used to implement four transversal

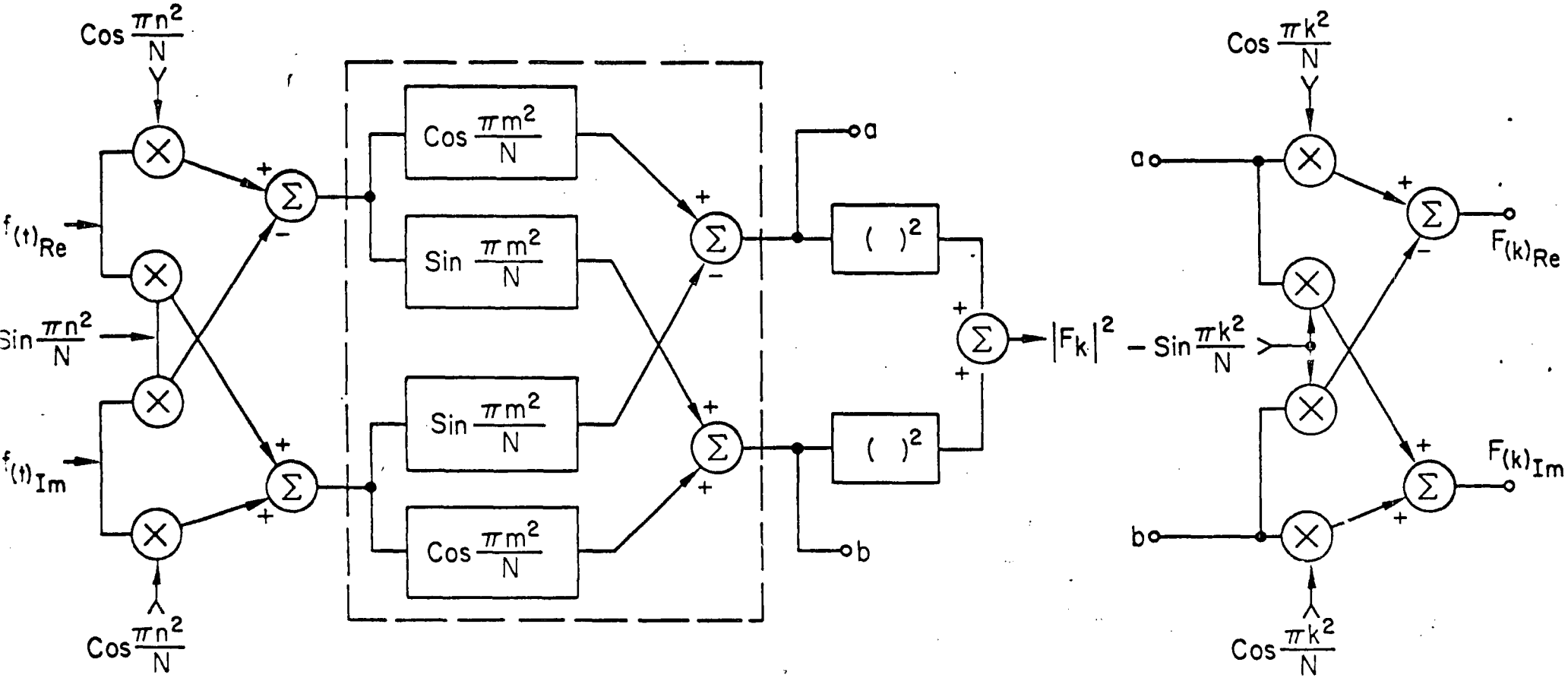


FIGURE 2.7. IMPLEMENTATION OF CHIRP-Z TRANSFORM ALGORITHM.

filters using a split-electrode weighting technique (31). The filter weighting coefficients and internal circuit connections are configured so that the device, in conjunction with additional off-chip components, can implement the CZT algorithm to calculate a 512-point DFT (32,33).

An evaluation module containing the R5601 device was available, which included additional circuitry necessary to compute the power spectrum of an analogue input signal. No phase information is obtainable with this module, as the post-multiplier unit is replaced with a hypotenuse function which recovers the spectral amplitude from the component cosine and sine terms. From (2.3), the squared spectral amplitude of a sequence x(n) can be expressed as:

$$\left| X(k) \right|^2 = \left| \sum_{n=0}^{N-1} x(n)\ e^{-j\pi n^2/N}\ e^{j\pi(k-n)^2/N} \right|^2 \qquad (2.4)$$

The final phase multiplier term, $e^{-j\pi k^2/N}$ has unit magnitude and has therefore been omitted from the above expression. The input data is stepped each time a new spectral component is calculated. Equation (2.4) then becomes:

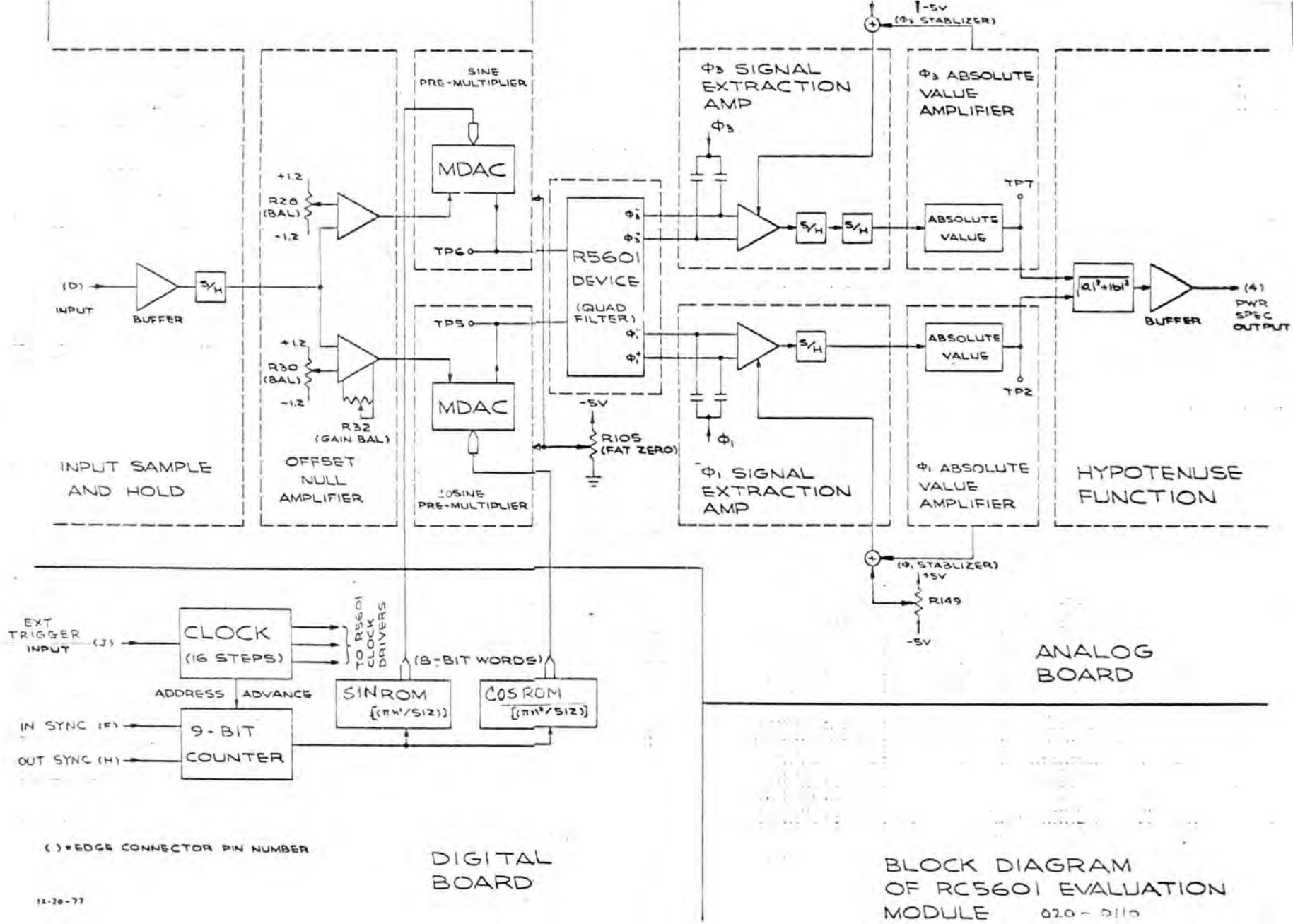
$$\left| X_s(k) \right|^2 = \left| \sum_{n=0}^{N-1} x(n+k) e^{-j\pi n^2/N}\ e^{j\pi(k-n)^2/N} \right|^2$$

The notation $X_s(k)$ indicates a "sliding" CZT.

A futher simplification is possible if the input is purely real, as it is in this case. The imaginary input is always zero so that two of the input multipliers may be deleted and the input

circuit simplified. A block diagram of the evaluation module is shown in figure 2.8. The real (analogue) input signal is buffered and converted to discrete-time samples by the input sample-and-hold, then split into the direct and quadrature channels. The sample values are multiplied by the appropriate chirped waveform using multiplying digital-to-analogue converters. The digital inputs to these converters are derived from two 512-by-8 bit ROMs which contain the sampled chirped sine and cosine waveforms. The sampled analogue products are then used for input to the R5601 four- channel convolution filter. Outputs from the filter are sampled and held to give time coincidence of all outputs, and then combined on an rms basis to give the spectral density of the input waveform.

Four clock phases are required by the filter device to propagate the discrete signal packets through the CCD channels. These are designated T1CL, T2CL, T3CL and T4CL, and are generated by a multi-phase clock generator circuit incorporated in the evaluation module which may be driven either from a 1 MHz internal oscillator or from an external trigger source. The sample rate with the on-board oscillator is a nominal 100 kHz, but lower rates are attainable with external triggering. The "address advance" pulse increments a 9 bit counter which addresses the weighting factor PROMs.



BLOCK DIAGRAM OF RC5601 EVALUATION MODULE 020-0110

FIGURE 2.8.

2.10.1 Post-multiplier unit

A post-multiplier unit was designed and constructed, enabling the real and imaginary components of the signal spectrum to be determined. The weighting factors for the algorithm illustrated in figure 2.7 are the sine and cosine chirp signals $e^{-j\pi k^2/N}$. However, because the transform is sliding and not stationary, the coefficients $X_s(k)_{Re}$ and $X_s(k)_{Im}$ do not provide a true measure of the input signal phase. A correction factor must be applied if the phase is to be restored. Consider the examples illustrated in figure 2.9 where the 8-point DFT's are evaluated of two sinusoids of frequencies $4.\pi n/N$ (solid line), and $2.\pi n/N$ (broken line).

The stationary DFT of an input sample sequence {x(n)} is derived from the time samples x(i), $0 \leq i \leq N-1$. The stationary DFT of the first signal in the example results in an output spectrum of -4j at k=2 and +4j at k=6, while that of the latter results in an output spectrum of similar coefficients at k=1 and k=7. The sliding DFT, however, is derived from time samples x(i), $n+k \leq i \leq N-1+k$. The sliding DFT results in an output spectrum of +4j (k=2) and -4j (k=6) for the first signal, and ($\sqrt{8}$ - $\sqrt{8}$j) (k=1) and ($\sqrt{8}$j - $\sqrt{8}$) (k=7) for the latter. Although the magnitude of the power spectrum is the same as for the stationary DFT, the sliding transform imposes an additional phase shift of $2.\pi k^2/N$ radians, which must be corrected for by multiplying the output spectrum by $e^{-j2.\pi k^2/N}$.

It has been stated that, in the case of the CZT implementation, the post-multiplication coefficients are the chirp

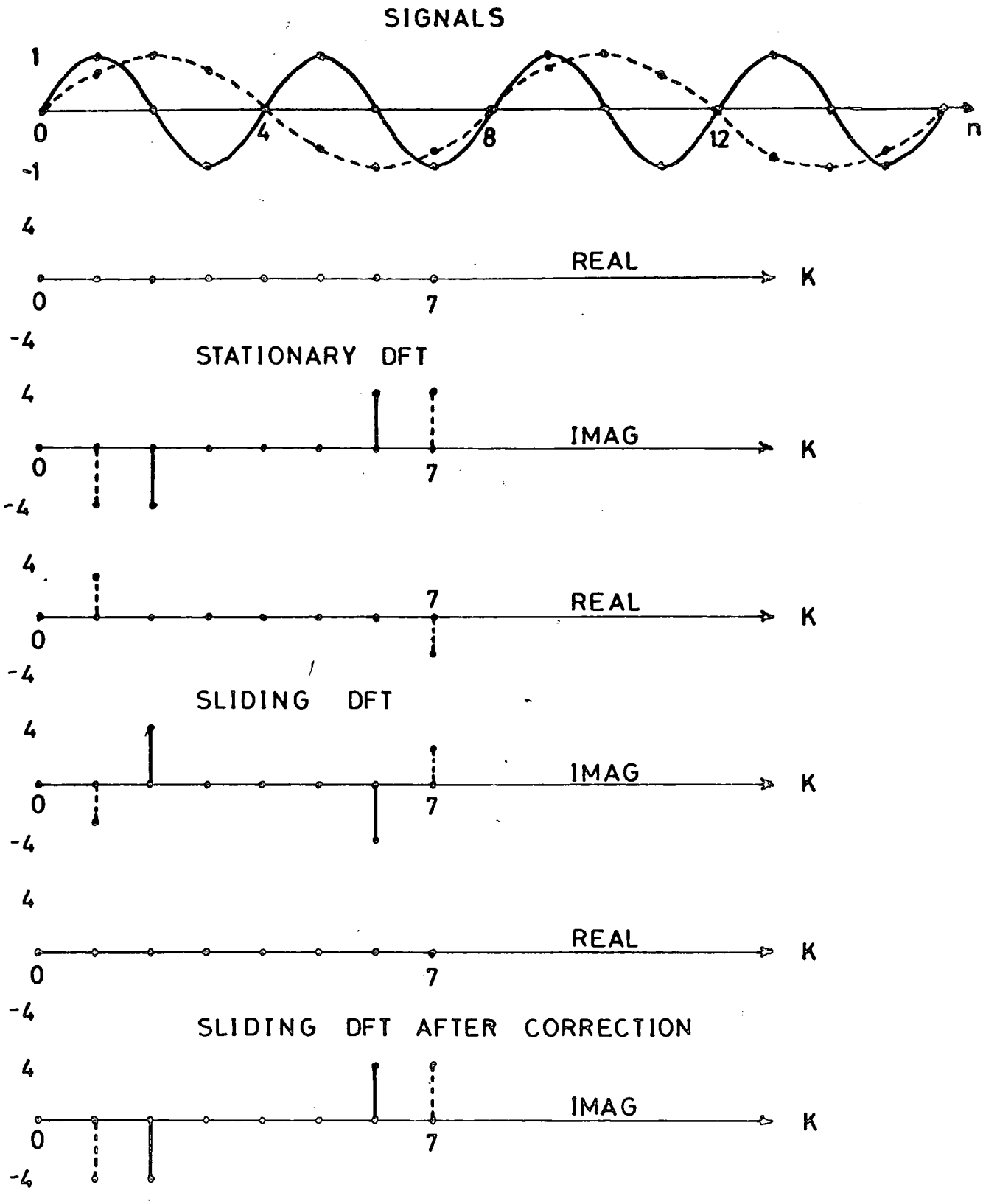


FIGURE 29 SLIDING PHASE CORRECTION.

waveforms given by $e^{-j.\pi k^2/N}$. For the sliding CZT implementation, the overall post-multiplication coefficients must be the product of these chirp waveforms and the phase correction factor. ie.

$$e^{-j\pi k^2/N}.e^{-j2\pi k^2/N} = e^{-j3\pi k^2/N}$$

2.10.2 Post-multiplier design

A circuit was required which could evaluate the complex multiplication $(a+jb)e^{-j3\pi k^2/N}$.The coefficients a and b are the direct and quadrature channels respectively from the transversal filter and are available from the evaluation module as sampled analogue voltages. The cosine and sine chirp waveforms could be stored in PROMs as 8-bit digital words. The transform length was fixed by the filter to be 512 points and each PROM was therefore required to have a 512-byte capacity. It was decided to use Intel 2716 type 2 kbyte EPROMs for two reasons: these devices were readily available at low cost, and a programming facility was already installed in the MSI 6800 development system. Two such EPROMs were programmed with the required chirp waveforms, which were generated using the co-resident BASIC interpreter and converted to offset binary format prior to writing to the device. Four real multiplications and two real additions were required to evaluate the complex multiplication described above. Each multiplication involved finding the product of a sampled bipolar analogue voltage and an 8-bit digital word. A circuit for performing this operation is shown in figure 2.10 and is described as follows:

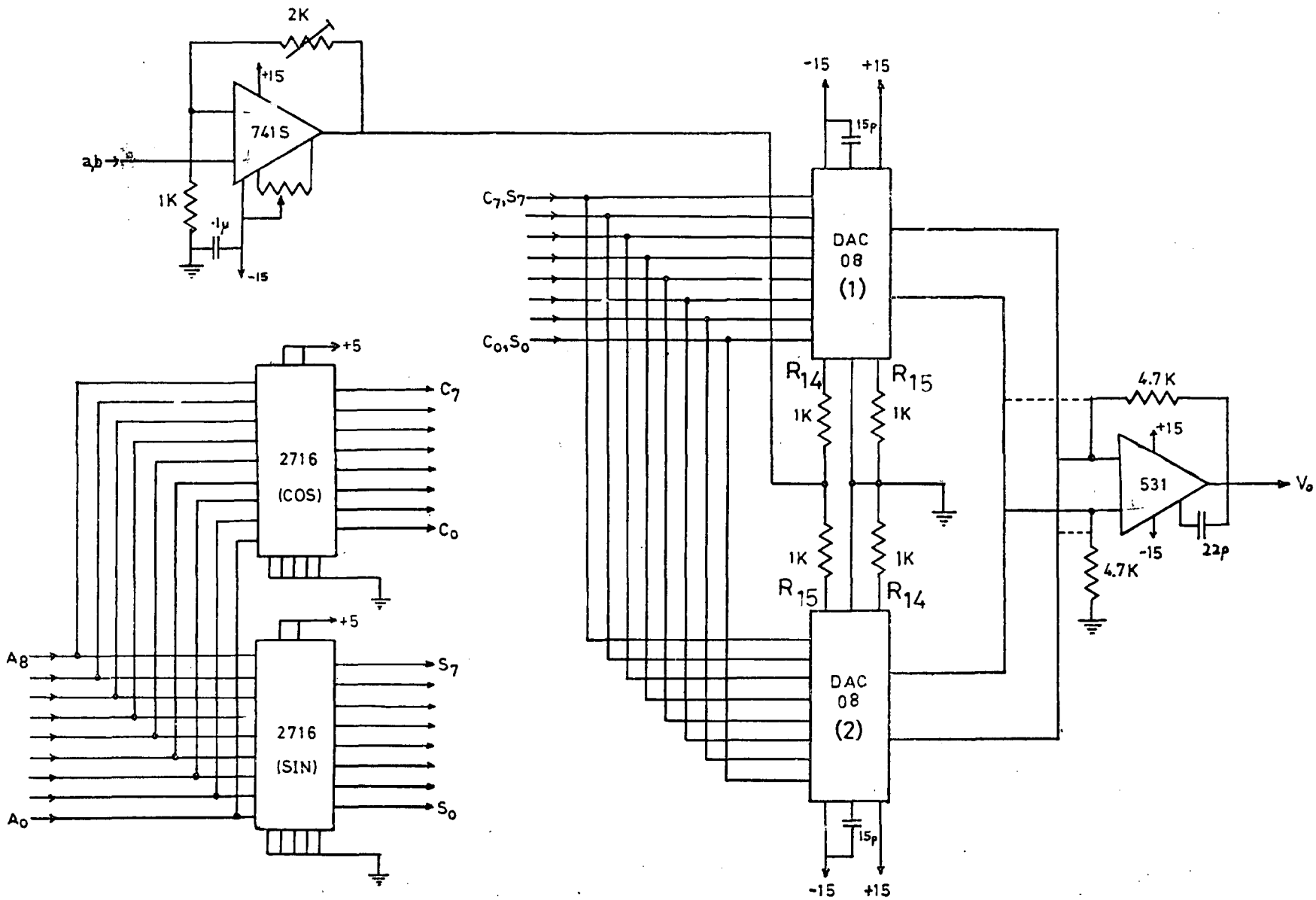


FIGURE 2.10(a). POST-MULTIPLIER UNIT (EPROMs and MDACs).

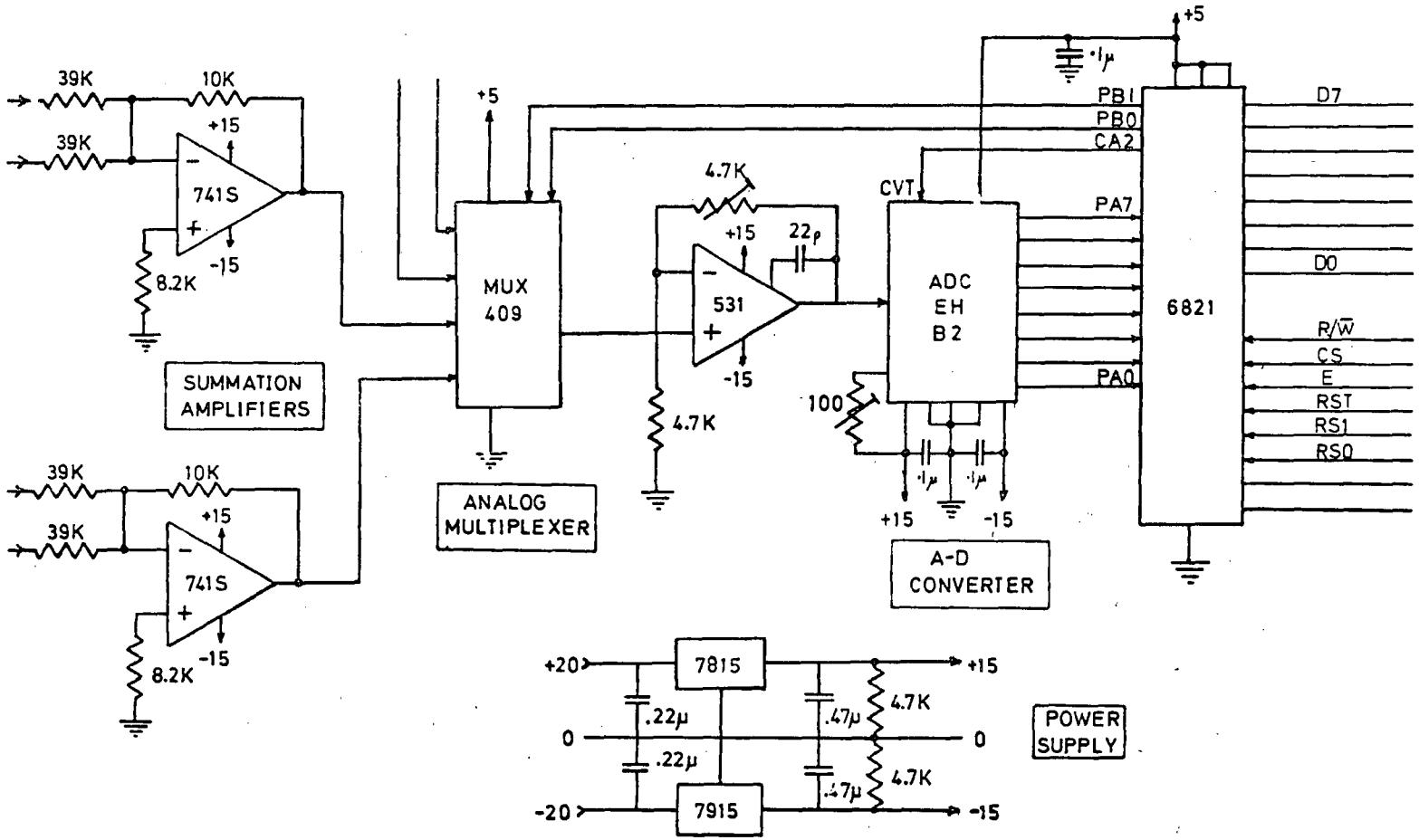


FIGURE 2.10(b). POST-MULTIPLIER UNIT (Summation amps, multiplexer, A-D converter).

The DAC-08 is an inexpensive monolithic digital-to-analogue converter with an 85ns settling time, which provides an output current which is the product of an 8-bit binary word and an input reference current, I_{REF}. The output current, I_o, for a binary input word, W, is given by:

$$I_o = \frac{256}{W} \times I_{REF}$$

Two complementary current outputs are available, which allow the device to be connected in a "symmetrical offset binary" output configuration by converting the currents to voltages and summing using an operational amplifier. Full scale positive and negative voltage outputs are then obtained for input words of $FF and $00 respectively. The output voltage is proportional to (W-128), where W is the input word.

Two input configurations are possible to cater for positive or negative reference voltages. For the circuit of figure 2.10(a), current flows into pin 14 of DAC 1 when V_{IN} is positive, but no reference current is available to DAC 2. If V_{IN} is 1V, the reference current is 1mA and the output voltage from the buffer amplifier is -4.7V x W. If V_{IN} is negative, current flows in to pin 14 of DAC 2 and the output voltage is positive. The output voltage is therefore given by

$$V_o = + \frac{4.7\ (W-128)}{128} \times V_{IN}$$

Table 2.1 shows the experimentally obtained output voltages for 5 analogue input voltages and 4 digital words presented to the multiplier from a PIA attached to the 6800 development system.

	digital word				
	V_i	$00	$7F	$80	$FF
input voltage	+1V	-4.79V	-0.01V	0.01V	4.79V
	+0.5V	-2.49V	-0.01V	0.01V	2.49V
	0	0	0	0	0
	-0.5V	2.49V	0.01V	-0.01V	-2.49V
	-1V	4.79V	0.01V	-0.01V	-4.79V

Table 2.1

The peak voltages at a and b on the evaluation board were measured for a fixed frequency sinusoidal input and were found to be ± 0.5V. The two non-inverting input buffer amplifiers on the post-multiplier board were arranged to have adjustable gains of between 1 and 3 in order to provide suitable variable references to the multipliers.

The eight output lines from the cosine chirp EPROM were connected to two multiplier circuits to give the products $-a.\cos 3\pi k^2/N$ (V_{TP3}) and $-b.\cos 3\pi k^2/N$ (V_{TP6}) and those from the sine chirp EPROM were connected to multiplier circuits to provide $a.\sin 3\pi k^2/N$ (V_{TP4}) and $-b.\sin 3\pi k^2/N$ (V_{TP5}). The latter product was obtained by using the output buffer amplifier in the non-inverting configuration. The two real products were summed in a wideband summing amplifier to give X(k) (real) and the two imaginary products were summed to provide X(k) (imaginary). Using the component values shown in the circuit diagram, the output voltages were

$$v_o(\text{real}) = -(10/39).(v_{TP3} + v_{TP5})$$

$$v_o(\text{imag}) = -(10/39).(v_{TP4} + v_{TP6})$$

The output voltage range in each case was ± 4.8V.

The summing amplifier outputs were connected to two inputs of a CMOS analogue multiplexer, the output of which was connected via a variable gain buffer amplifier to a high speed (2μs conversion time) bipolar input 8-bit analogue-to-digital converter. Interface to the 6800 microprocessor system was via a single PIA. Four pairs of multiplying D-A converters, as in figure 2.10(a) were necessary to evaluate the four real multiplications required to find the complex product.

The complete circuit was constructed using wirewrap techniques on a 6.5 x 4.4 ins. circuit board. The ±20v supply to the RC5601 evaluation board was reduced to ±15v to supply the post-multiplier by using one 7815 and one 7915 voltage regulator IC. A photograph of the board together with the CCD evaluation board can be seen in photo 2.1.

2.10.3 Test Results

A series of tests were performed on the post-multiplier board and the results compared with theoretical predictions to ensure that the circuit was functioning correctly. The sine and cosine chirp waveforms generated by the unit are shown in photos 2.2 and 2.3. The EPROMs were addressed directly from a PIA installed in the 6800 development system and the digital outputs from the A-D converter were read for a variety of DC analogue inputs and a variety of weighting factors. The results of these tests are shown in table 2.2. a and b are analogue input voltages, ADDR is the EPROM address, COS and SIN are the cosine and sine outputs from the EPROMs. R and I are the real and quadrature output voltages respectively ("ex" and "th" indicate experimental and

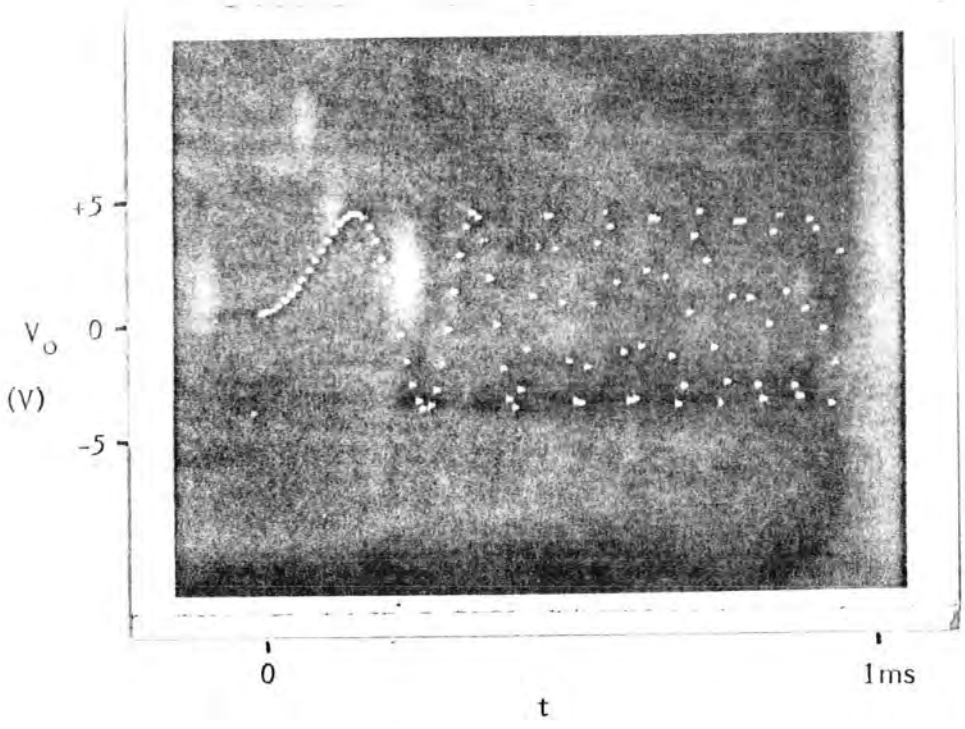


PHOTO 2.2. SINE CHIRP WAVEFORM

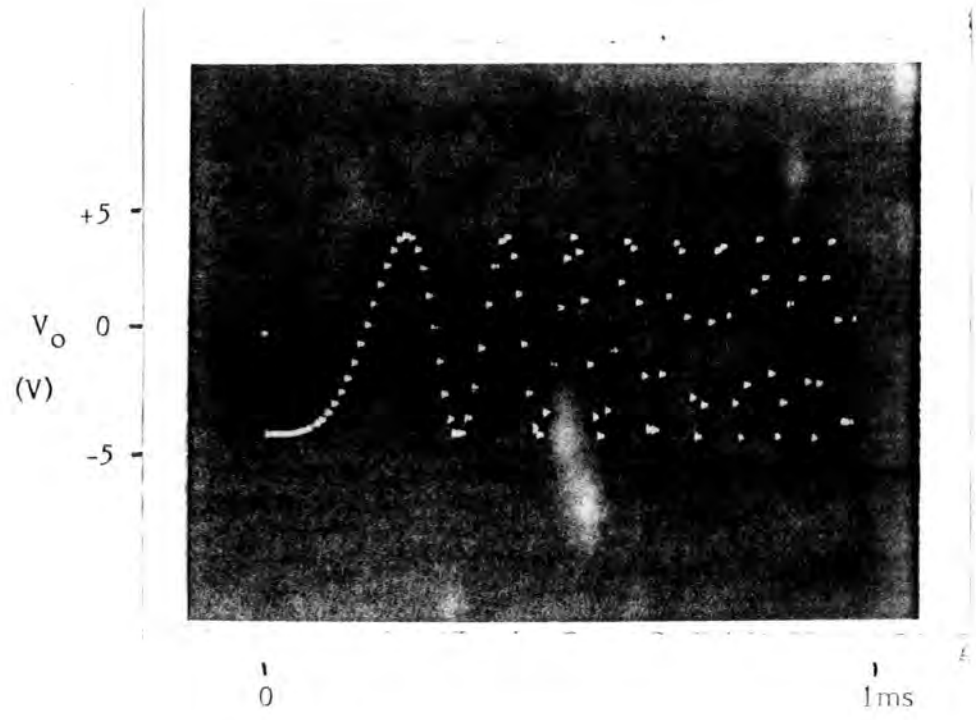


PHOTO 2.3. COSINE CHIRP WAVEFORM.

theory respectively).

a	b	COS	SIN	R(ex)	R(th)	I(ex)	I(th)	ADDR
-0.5	-0.5	$40	$C0	1.18	1.175	0	0	$08
+0.5	+0.5	$C0	$40	1.18	1.175	0	0	$10
-1.0	-1.0	$7F	$FF	2.37	2.35	-2.36	-2.35	$0E
-1.0	+1.0	$00	$FF	0	0	-4.71	-4.70	$04
+1.0	-1.0	$FF	$7F	2.35	2.35	-2.36	-2.35	$16
-1.0	-0.5	$FF	$C0	-1.75	-1.76	-2.73	-2.75	$17
-0.5	+1.0	$00	$00	3.54	3.53	-1.19	-1.18	$00
-0.5	+0.5	$FF	$00	0.01	0	2.36	2.35	$14

Table 2.2

The CCD evaluation module was interfaced to the microprocessor system and to the post-multiplier unit as illustrated in figure 2.11. The 9 address lines from the on-board counter were connected to the post- multiplier EPROMs, and the "a" and "b" outputs from the filter output buffers were connected to the real and quadrature inputs respectively of the post-multiplier system. The $\overline{\text{T3CL}}$ clock line was connected to the CA1 control line of the PIA and the "in sync" input was derived from the CA2 line. The CA2 line was also connected to the "gate enable" input of a pulse generator having a gated output; the latter set to deliver 500 ns wide pulses at a repetition frequency of 20 kHz into the "ext trig" input of the evaluation board. This enabled the system to operate at a sample rate of 20 kHz. The CB2 control line was configured such that a "write to B side data" instruction causes a short pulse to occur immediately after the instruction. This pulse was used to initiate an A-D conversion.

The sinusoidal output from a function generator was connected to a frequency counter and to a comparator, whose output was

clamped with a 5.1V zener diode and fed to the PB7 line of the post-multiplier PIA.

Two programs were written to test the functioning of the post-multiplier unit. The first was not synchronised to the input signal, and the second was synchronised to the zero crossings of the input signal, as detected by the comparator. A description of the operation of the two programs is as follows.

CA2 is initialised to '0', which disables the pulse generator output and holds the on-board 9-bit memory address counter in "reset". A counter within the microprocessor system is then initialised to a value of 512, and the CA2 line is brought to a logic '1', thereby beginning the sampling process. Each transition of $\overline{T3CL}$ causes an interrupt to be generated which is used to decrement the microprocessor counter. After 512 interrupts have been counted, this counter is set to zero and the CCD filter is full (it is a 512-stage device). During the next 256 transitions of $\overline{T3CL}$, the CCD evaluation module outputs frequency samples in the Nyquist band; therefore another 256 interrupts are counted. The real and imaginary outputs from the post-multiplier unit are then read by the microprocessor system using the analogue multiplexer and the A-D converter. The results are stored for further processing.

The sampling frequency was set to 20 kHz and a sinusoidal input of approximately 4.5 kHz was applied to the CCD evaluation module. Sampling was initiated at random intervals and the results were stored in two 256 byte tables in memory. An examination of the results showed that the energy associated with a single

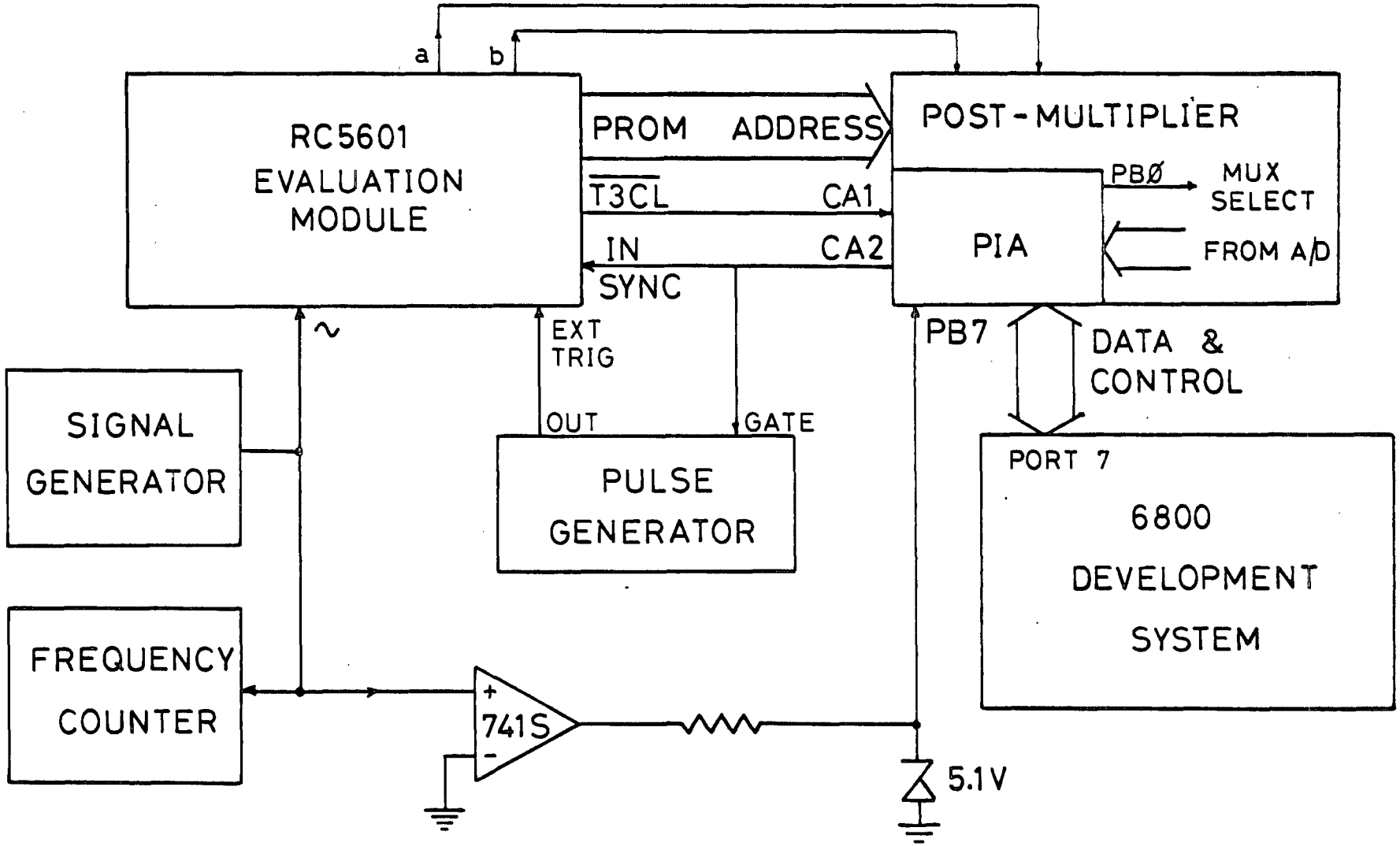


FIGURE 2.11 POST-MULTIPLIER TEST CIRCUIT

frequency carrier tended to be spread over several frequency bins due to the effect of the Hanning window applied to the CCD split electrode weightings.

The experiment was repeated and the resulting coefficients around the frequency of interest were printed on the terminal (in decimal) on each occasion. Figure 2.12(a) shows the results obtained from successive readings of the real and imaginary coefficients. Five pairs of coefficients are printed for each reading; the centre coefficient is that corresponding to the frequency bin of the input signal.

Next, the spectrum analyser was synchronised to the incoming signal by initiating the sampling process only on a positive transition of PB7 (corresponding to a zero-crossing of the input data signal), so as to obtain samples of a sinusoid with zero phase shift. The real and quadrature coefficient magnitudes around the frequency of interest were printed as before, and the experiment was repeated a number of times. A correlation was observed between successive results, as shown in figure 2.12(b).

The differential inputs to the comparator were then reversed, and the experiment was repeated, without varying the magnitude or the frequency of the input signal, to obtain samples of a sinusoid with 180^{o} of phase shift (a negative sinusoid). An examination of the output coefficients around the frequency of interest revealed that the signs of the coefficients were reversed, although the magnitude of the spectrum remained constant. In fact, the coefficients for the first case were predominantly

NOT SYNCHRONISED:

REAL					IMAG				
-2	-26	-64	-31	-1	-1	-9	-21	-33	0
0	12	43	41	0	-1	-24	-51	-19	0
2	24	53	18	0	0	8	21	24	1
0	-4	8	27	0	0	-24	-65	-40	-2
0	25	63	39	3	0	-4	-9	-7	0
0	-23	-59	-27	0	0	-5	-30	-39	0
0	-9	-6	18	0	0	-22	-54	-45	-2
2	23	53	33	1	0	-3	-9	-4	0
0	2	11	8	0	1	10	23	8	0
0	-7	-10	3	0	0	15	32	54	2
1	13	45	2	0	0	-2	-33	-25	0

FIGURE 2.12(A). TEST OUTPUT DATA AT K=114,115, . . 118

SYNCHRONISED TO INPUT:

REAL					IMAG				
0	-4	8	10	0	0	-27	-53	-35	0
0	-5	6	11	0	0	-25	-53	-8	0
0	-4	9	8	0	0	-23	-57	-30	0
0	-4	7	11	0	0	-23	-59	-32	0
0	-6	1	7	0	0	-21	-56	-33	0
0	-3	2	9	0	0	-26	-53	-32	0
0	-3	2	7	0	0	-24	-51	-31	0
0	-5	3	7	0	0	-26	-52	-30	0
0	-4	3	9	0	0	-25	-53	-29	0
0	-4	7	7	0	0	-25	-57	-31	0
0	-1	0	3	0	0	-33	-59	-34	0

FIGURE 2.12(B). TEST OUTPUT DATA AT K=114,115, . . 118
SAMPLING SYNCHRONISED TO INPUT SIGNAL.

imaginary and negative and for the second case were predominantly imaginary and positive. These results corresponded to the spectrum obtained from samples of a sinusoid and a negative sinusoid respectively, although several unwanted components (with incorrect signs) were also present due to the effect of the Hanning window weighting of the filters.

Several further tests were carried out on the post-multiplier unit to confirm that it was functioning correctly. By using the analogue multiplexer it was possible to select (using software) the real coefficient, the imaginary coefficient (and hence the phase), or the power spectrum, for each frequency bin.

2.11 Multiply-Divide Unit

The hardware multiply-divide unit (34) used in the software FFT implementations is now described. The RCA-CDP1855C is an 8-bit monolithic multiply/ divide unit (MDU) which performs multiplication and division operations on unsigned, binary operators. It is based on a method of multiplying by add and shift right operations and dividing by subtract and shift left operations. The device is structured to permit cascading of identical units to handle operations up to 32 bits.

Two MDU's were cascaded to permit the following operations:

(i) a 32-bit by 16-bit divide yielding a 16-bit result and a 16-bit remainder.

(ii) a 16-bit by 16-bit multiplication yielding a 32-bit result

Each MDU has 8 bi-directional tri-state data lines, a read/$\overline{\text{write}}$ input, a clock input, a reset input, a chip enable input and 3 register select lines. Interface to the MSI 6800 system was provided via I/O port 7 on the system I/O bus. The chip enable signal from the bus was inverted and connected to the CE and RA2 lines on the MDU's. The data lines, the reset line and the R/$\overline{\text{W}}$ line from the bus were connected directly to the corresponding pins on the MDU's. RS0 and RS1 were connected to RA0 and RA1 respectively. The addresses of the MDU registers were then as follows:

$F538	X
$F539	Z
$F53A	Y
$F53B	Status (R/$\overline{\text{W}}$ = 1)

The circuit diagram of the cascaded MDU system is shown in figure 2.13. Interconnections were made using wirewrap on a 0.1in. matrix circuit board.

For two cascaded MDU's each of the registers X,Y and Z is two bytes wide although each occupies only a single address. Each MDU contains a "sequence counter" which enables registers to be loaded or read sequentially. When the counter matches the chip number (CN1,CN0 lines) the device is selected. For example the first selection of register X for loading loads the most significant MDU, the second selection loads the least significant.

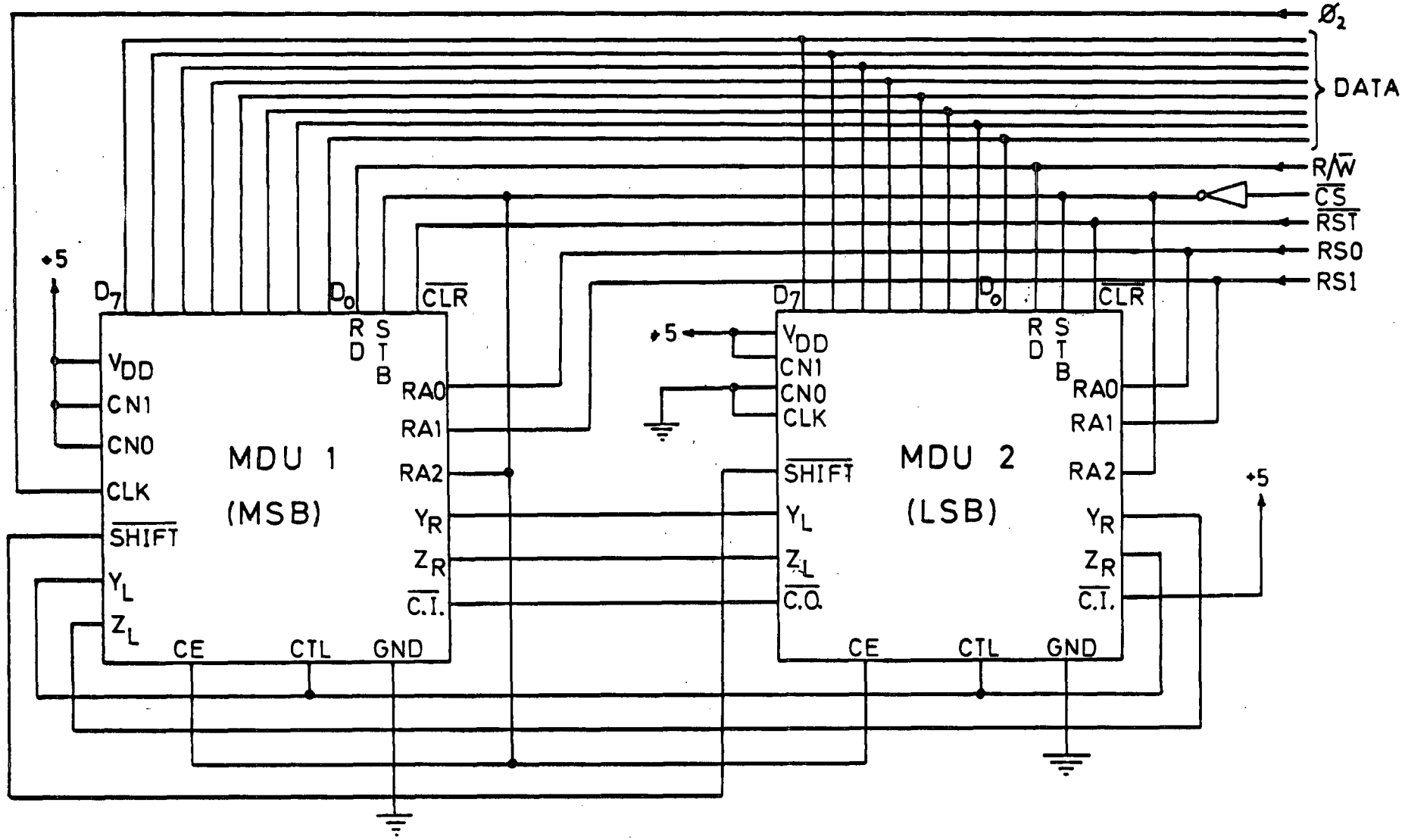


FIGURE 2.13. MULTIPLY-DIVIDE UNIT

To execute a multiply or a divide operation, the data is simply loaded into the appropriate registers, and a control word is written to define the required operation. The results of the operation can then be obtained by reading the appropriate registers.

If N MDU's are cascaded, all operations require 8N+1 shift pulses. For 2 cascaded devices operating at a frequency of 1 MHz therefore, the time taken for a multiplication or a division is 17 us. This is a substantial improvement over a purely software implementation, and was found to provide a considerable improvement in execution time for the Fast Fourier Transform algorithm implementation.

2.12 Comparison of algorithms

The four DFT implementations previously described were compared mainly from point of view of memory requirement and execution time. The first implementation (using a high-level language interpreter) was obviously poor in all respects except for ease of programming and shall not be considered further. The assembly language implementation was quite efficient in terms of memory utilisation and was two orders of magnitude times faster than the BASIC routine. More efficient memory use could have been obtained by computing the butterfly weighting factors only when required; however, this would have led to a considerable increase in execution time. The maximum permissible transform length was 256; if this length had been increased, many single-precision operations would have required double precision, again increasing the required time.

A considerable improvement in execution speed was obtained by using a hardware multiply-divide unit (described in the previous section). This unit was used to (a) multiply the input data sequence by a time window (if required), (b) perform all weighting factor multiplications during the butterfly computations, (c) compute the squares of the Fourier coefficients (if a power spectrum is required) and (d) compute the square root of the sum of the squares of the coefficients using the Newton-Raphson iteration formula (which requires division operations). Table 2.3 below shows figures for execution times for the various algorithms.

N	BASIC	ASSM(1)	ASSM(2)	CZT
8	1.2s	19ms	5.1ms	-
16	2s	54ms	14.2ms	-
32	4.7s	164ms	37.5ms	-
64	10s	429ms	96ms	-
128	22.8s	1.01s	219ms	-
256	48.3s	2.5s	501ms	-
512	105.2s	-	-	5.12ms

Table 2.3

ASSM(1) and ASSM(2) refer to the assembly level FFT routines with and without the hardware multiplier, respectively.

The hardware implementation of the CZT using the charge coupled device was by far the most efficient in terms of execution time. The device itself is primarily for use in power spectrum estimation, and the evaluation board contains peripheral circuitry to evaluate the power spectrum of a fixed-length (512 point) transform. It has been shown that it is possible to use additional circuitry to evaluate the Fourier coefficients, which may

subsequently be used to determine the phase of an input signal.

2.13 Conclusion

The microprocessor chosen for the work in this thesis has been introduced and some preliminary work has been demonstrated on the transfer of data to and from magnetic tapes using a novel software interface. The implementations of efficient algorithms for spectral analysis have been described and investigated. Phase information may be obtained from an analysis of the Fourier coefficients resulting from the Discrete Fourier Transform of an input time signal. This phase information may be utilised to implement a demodulation process in phase modulated data transmission systems for HF radio communications. However, the execution time involved for the algorithm computations may sometimes be excessive and a hardware approach may sometimes be more favourable. An investigation into Fourier transformation using charge coupled devices has been demonstrated which has led to the design and construction of a "post-multiplier" unit for determination of the Fourier coefficients under control of a microprocessor system. This work has formed the basis for a demodulator design for a multitone phase modulated signal for HF data transmission.

The spectral analysis techniques discussed in this chapter have been used extensively in applications described elswhere in this thesis.

CHAPTER 3 The HF Spectrogram

3.1 Introduction

It has been mentioned that the problems associated with digital data communications over HF radio links generally arise from one of two major causes:

(i) Multipath propagation caused by multiple reflections from the ionosphere.

(ii) Noise; originating from both natural and man-made sources.

The phenomenon of multipath propagation can give rise to intersymbol interference if the signal element duration is of insufficient length, and can also cause fading of the received signal resulting from destructive interference. The received signal may also be corrupted by noise, which may be broad- or narrow-band, and which can severely degrade the fidelity of the detected data.

Because of the predominance of analogue traffic over HF radio links, the 3kHz "voice" channel has become accepted as the standard communications channel for such a medium, and requests for frequency allocations are normally granted on condition that the radiated signal shall not occupy a bandwidth which exceeds this figure. This restriction applies even when the information is digital in nature and it therefore becomes the task of the communications engineer to design efficient systems for data links operating over such channels.

It is worthwhile observing the effects of fading and noise on a signal confined to a 3kHz bandwidth in order that the predominant disturbances may be identified. The results may yield clues which could be used to improve system performance by better design, or, if implemented in real time, a number of the available channels may be monitored, and the best chosen for subsequent transmission of data.

3.2 System principles

This chapter describes a system which provides a pictorial display of the time-varying spectral properties of a chosen voice channel. Frequency, on a linear scale from 0 to 3 kHz, is displayed along the y-axis, and time, from t=0 to a chosen value, is displayed along the x-axis. The intensity of the display at a particular pair of coordinates indicates the strength of a signal (or noise) which occurred at that particular time and at that particular spot frequency. Because of the digital nature of the system, both time and frequency are quantised, and the display actually consists of a two-dimensional array of points of varying intensity. However, these points can be arranged to be close enough together to provide a quasi-continuous pictorial representation of events occurring within the voice channel over a chosen time interval.

A similar technique was used by R.G.W. Thompson (36,37) to classify HF fading patterns. His analysis was based on observations of a multitone sounding signal from a remote transmitter, and the received data was recorded on an FM tape recorder prior to processing through a spectrum analyser and

mainframe computer. A display was presented on a facsimile receiver coupled to a digital to analogue converter. Disadvantages of the system were cost, complexity, and the time delay necessitated by the pre-recording of the received data. The microprocessor-based system described in this chapter is fast, versatile, and uses a storage oscilloscope to present the spectral information. Such a system could be developed into a small, economical unit suitable for portable operation.

A description of the operation of the system is presented, followed by details of the hardware and software employed in the implementation. Finally, some results are presented, together with suggestions for further development.

3.3 System implementation

Figure 3.1 shows a block diagram of the system. The audio output from an HF receiver is filtered, and then sampled by the A-D converter and microprocessor system. A set of N samples (N is a power of 2, $8 \leq N \leq 256$) is stored in RAM and the Fast Fourier Transform (FFT) algorithm is used to compute the power spectrum of the set which is displayed as a single vertical line on the storage oscilloscope. This line is quantised into N/2 discrete points and the intensity of the display at each point is a measure of the magnitude of the power spectrum at the corresponding frequency. The vertical spacing between each point is therefore 2h/N cms., where h is the height of the oscilloscope screen. The beam is returned to the x axis and is moved to the right by w/N_t cms., where w is the width of the screen and N_t ($8 \leq N_t \leq 256$) is the number of lines to be plotted. A new set

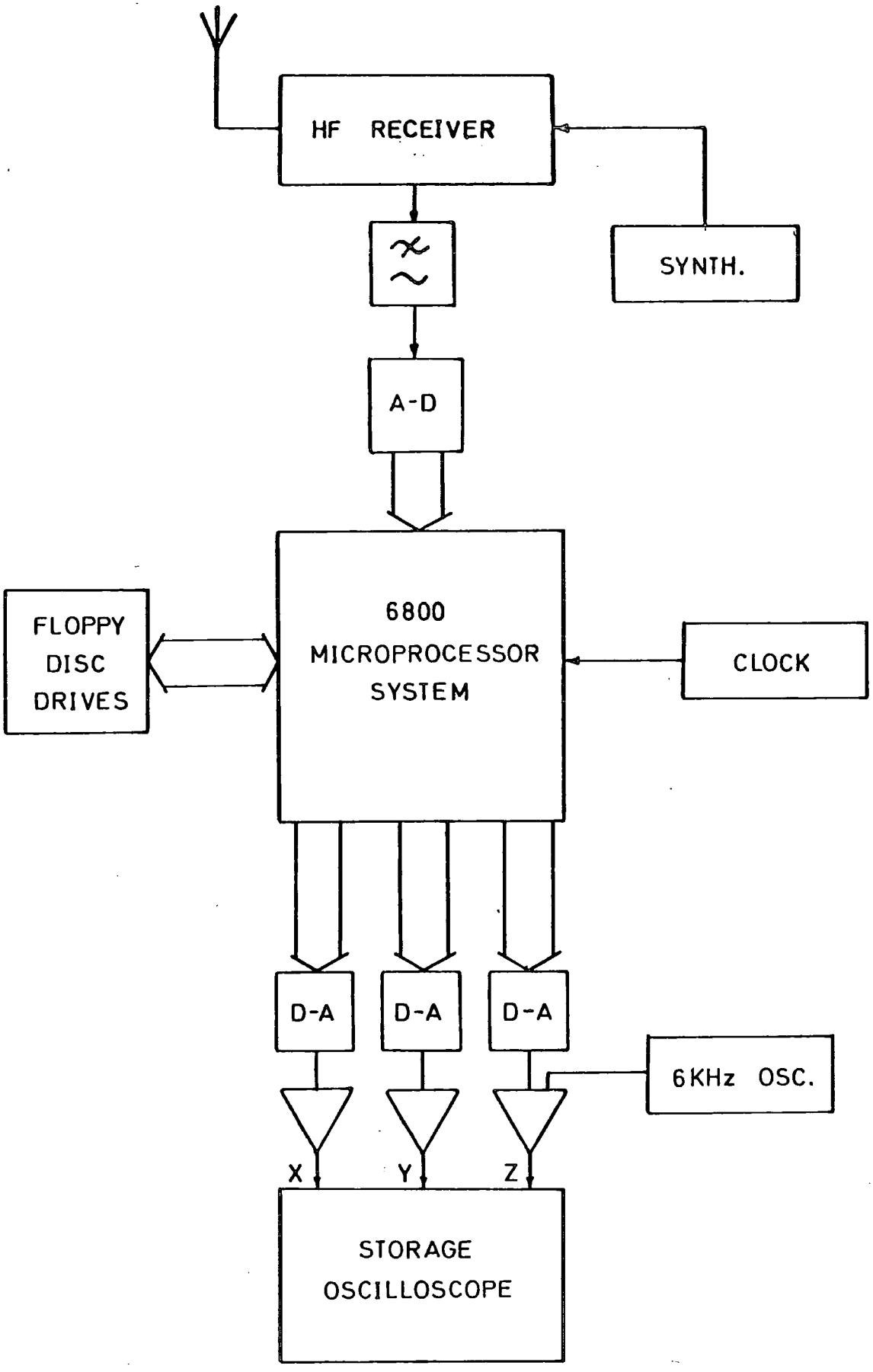


Figure 3.1 HF Spectrogram Block Diagram

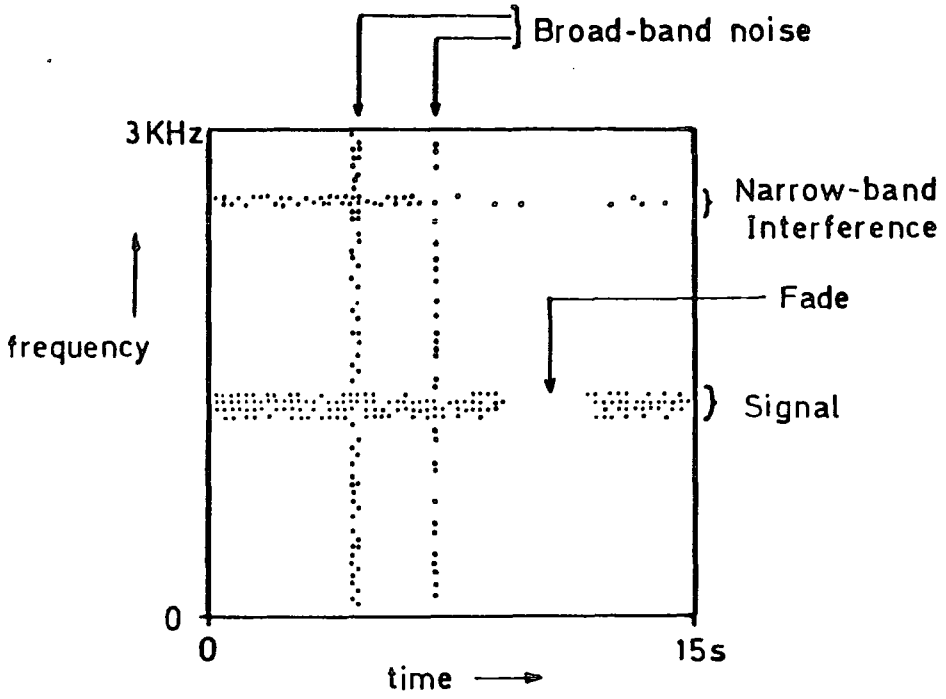


Figure 3.2 Example Display

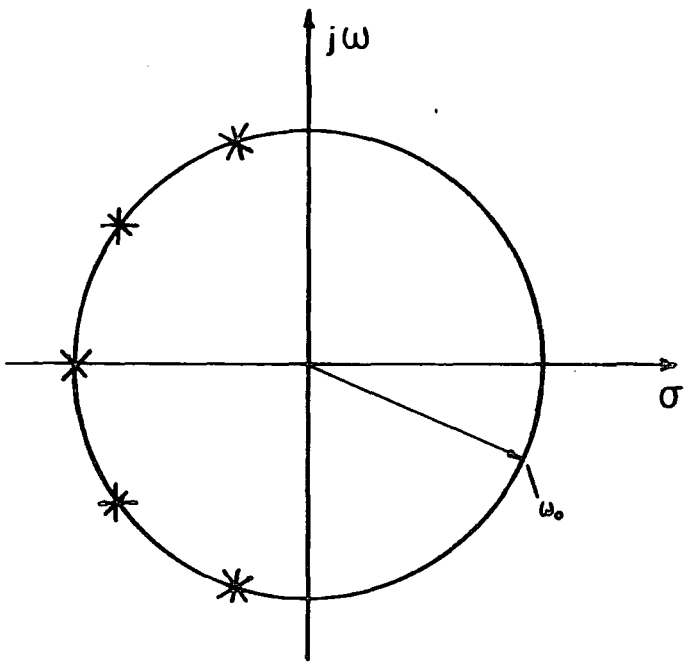


Figure 3.3 S-Plane Representation of Filter

of N samples is then acquired from the receiver and a second line is plotted. This procedure continues until the display is complete. An illustration of the kind of display which might be obtained is shown in figure 3.2.

3.3.1 The lowpass filter

The lowpass filter used in the system was required to have a cutoff frequency of 3 kHz to eliminate out-of-channel noise. The attenuation characteristic was designed to be uniform over the passband, and with a steep rolloff above the cutoff frequency. A two-pole filter would have been sufficient for this application; however a five-pole filter could be realised with little additional complexity and was found to be useful in a later application where a sharp cutoff was essential.

The general form for the squared-magnitude function of a Butterworth filter is given by:

$$\left| H(j\omega) \right|^2 = \frac{1}{1 + \left[\frac{\omega}{\omega_o}\right]^{2n}}$$

where n is the order of the filter and ω_o is the cutoff frequency. The s-plane pole locations which correspond to the denominator polynomial for a fifth-order filter are given by:

$$H(s).H(-s) = \frac{1}{1 + \left[\frac{s}{\omega_o}\right]^{2n}} \text{ (n even)}, \qquad \frac{1}{1 - \left[\frac{s}{\omega_o}\right]^{2n}} \text{ (n odd)}$$

This function has 10 poles equally spaced around a circle of radius ω_o in the s-plane; the n poles to the left of the imaginary axis define the filter (figure 3.3).

The design of the filter was based around the Datel universal hybrid active filter component, model FLT-U2. This dual-in-line package contains four operational amplifiers and a number of passive components. The first three amplifiers are "committed" in the sense that they are internally interconnected with a number of resistors and capacitors in such a way that it is possible to implement a second-order transfer function using the state-variable active filter principle, with the addition of a few external components. The fourth "uncommitted" op amp can be used as a buffer, or to add an independent real pole to the filter characteristics.

For this application, two filter units were cascaded to realise the five-pole Butterworth filter. Each trio of committed amplifiers was used to provide two poles, the uncommitted amplifier in the first unit was used to provide a buffer between the two units and the final, uncommitted, amplifier was used to provide the remaining (real) pole.

The conjugate poles P_1 and P_2 were implemented with the first filter unit.

$$\text{damping factor, } d_1 = \cos \phi_1 = 0.309$$

$$Q_1 = \frac{1}{2d_1} = 1.618$$

Appropriate components were chosen by consulting the manufacturer's data sheet for the FLT-U2. Two sets of tables are provided, one each for the inverting and non-inverting filter configurations. The first filter was operated in the non-inverting

mode and Table 3.1 shows the recommended, calculated, and actual values used in the implementation:

	recommended	calculated	actual
R_{11}	∞	∞	∞
R_{12}	316k/Q	195k	180k
R_{13}	100k/(3.16Q-1)	24.3k	24k
R_{14}	$5.03 \times 10^7/f_o$	16.7k	16k
R_{15}	$5.03 \times 10^7/f_o$	16.7k	16k

Table 3.1.

The uncommitted operational amplifier was wired as a unity-gain non-inverting amplifier to be used as a buffer between the two filter units.

The conjugate poles P_2 and P_4 were implemented using the second filter unit.

$$\text{damping factor, } d_2 = \cos \phi_2 = 0.809$$

$$Q_2 = \frac{1}{2d_2} = 0.618$$

Unit 2 was operated in the inverting mode and Table 3.2 shows the recommended, calculated, and actual component values used in the implementation.

	recommended	calculated	actual
R_{21}	100k	100k	100k
R_{22}	∞	∞	∞
R_{23}	100k/(3.16Q-1)	75k	75k
R_{24}	$5.03 \times 10^7/f_o$	16.7k	16k
R_{25}	$5.03 \times 10^7/f_o$	16.7k	16k

Table 3.2.

The real pole, P_3, was implemented using the remaining "uncommitted" op amp. A gain of -1 was required, which defined $R_7/R_6 = 1$. This pole was set to 3kHz by using the capacitor, C, across the feedback resistor R_7.

$$C = \frac{1}{2\pi f R_7}$$

Suitable values were found to be C = 4700pF and R_7 (= R_6) = 11kΩ.

A circuit diagram of the complete filter is shown in figure 3.4. The filter was constructed using a piece of copper strip board, and each of the two cascaded units was tested separately before testing the complete design. The plot of figure 3.5 shows the results obtained for each unit together with the overall response.

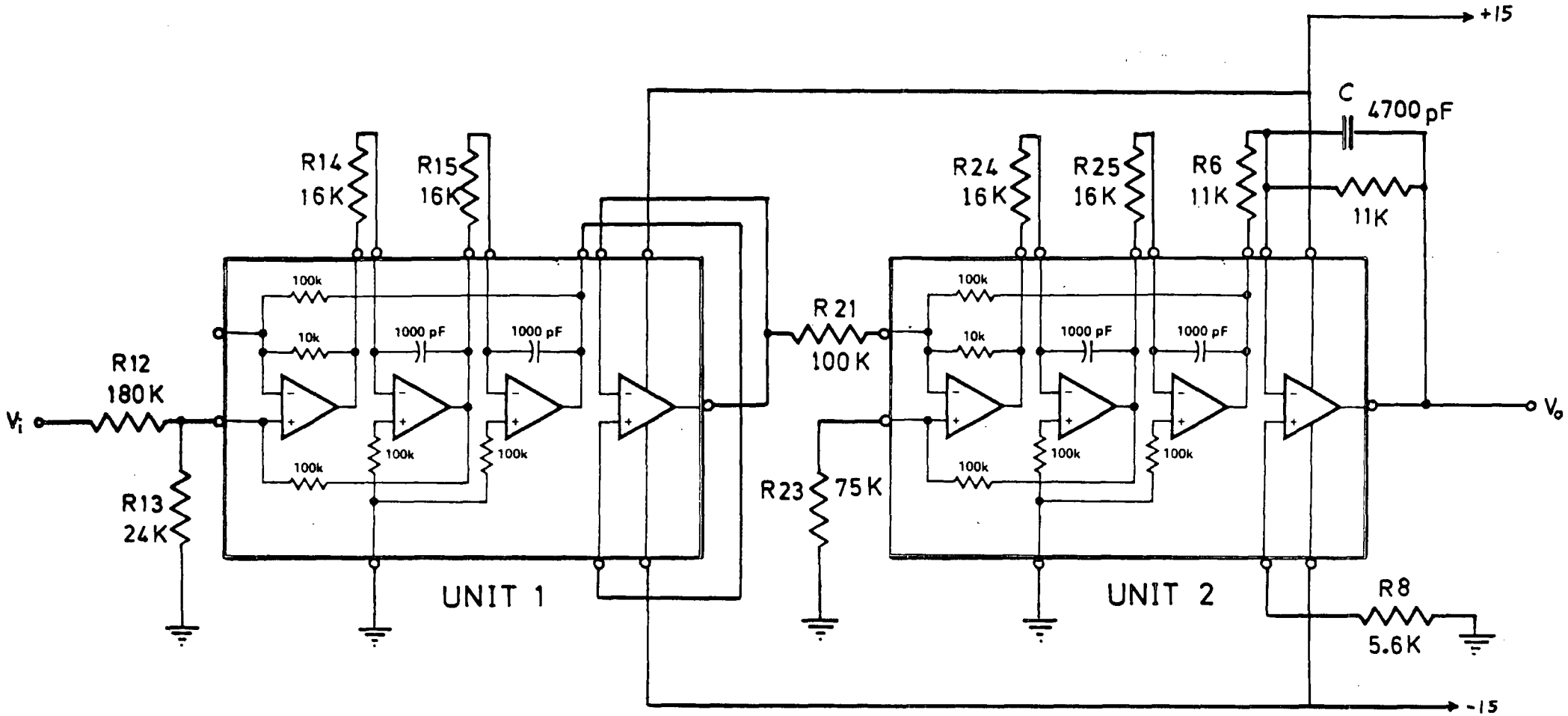


FIGURE 3.4 FILTER CIRCUIT DIAGRAM.

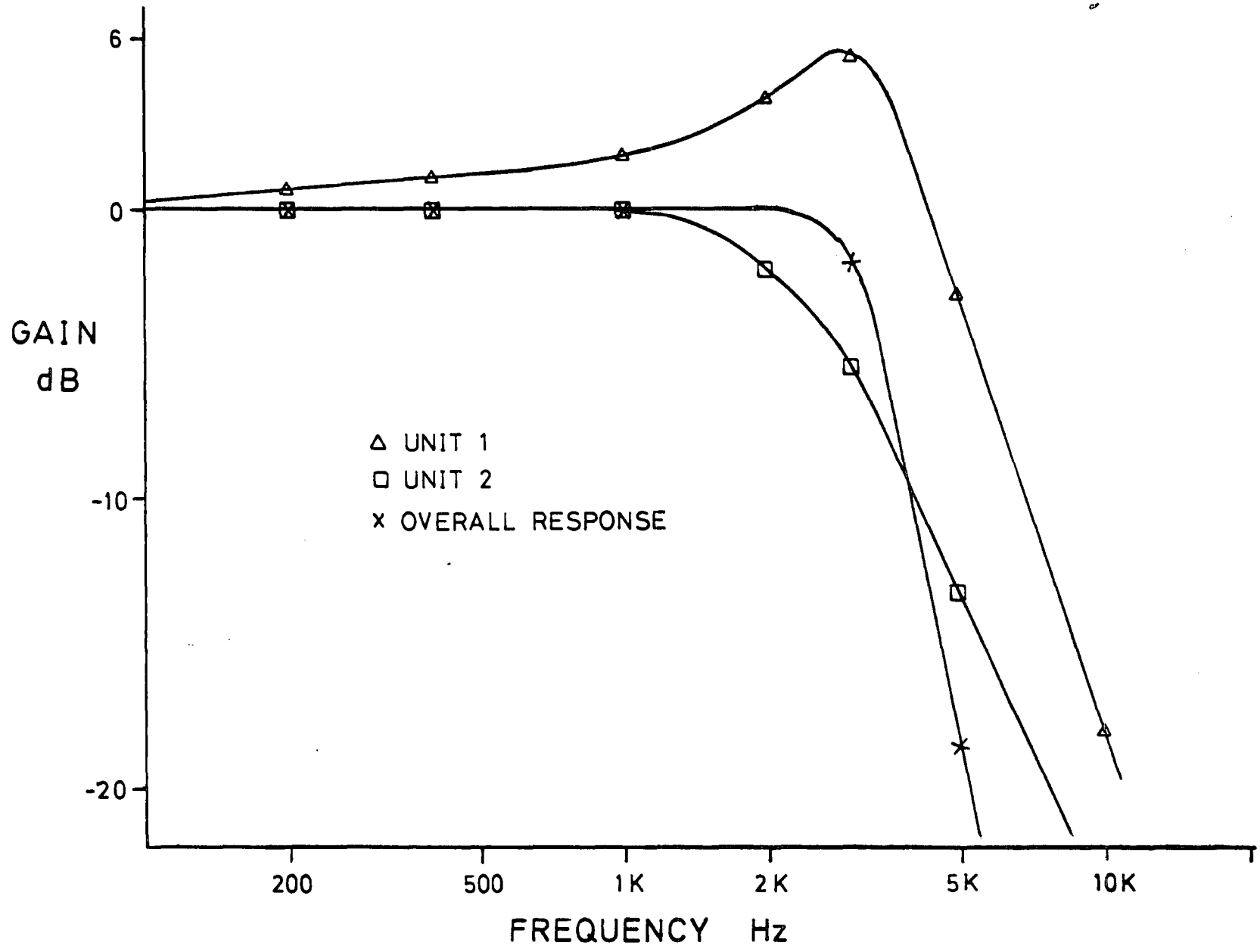


FIGURE 3.5. FILTER MAGNITUDE - FREQUENCY RESPONSE.

3.3.2 The analogue to digital converter

An 8-bit analogue-to-digital converter was required having a conversion time of 166.6 μs or less. This figure is the conversion time required for a sampling rate of 6kHz, which is twice the highest frequency component of interest. The conversion system was based around a component (the ZN425E) containing an 8-bit A to D converter together with an 8-bit binary counter, allowing the construction of a successive approximation A to D conversion system with the addition of an external voltage comparator. The clock input to the counter was provided by the $\emptyset_2$ clock of the microprocessor system. For a 1.8MHz clock, this allowed a conversion time of $2^8 / 1.8 \times 10^6 = 142$ μs, which was adequate for the purpose.

Interface to the microprocessor system was provided by a 6821 Peripheral Interface Adapter (PIA) which has two 8-bit peripheral ports which may be software configured for either input or output. The 8-bit digital output from the converter was connected to the 'B' port of this device and the CB2 control line was used to provide the 'convert' command. In order to fully utilise the I/O capabilities of the PIA, it was decided to construct a D-A converter on the same board connected to the 'A' side of the PIA which could then be used to provide the control signal for the oscilloscope Z modulation. A second ZN425E and an operational amplifier were used to construct the D to A system.

A complete circuit diagram of the A-D/ D-A system is shown in figure 3.6. The system was constructed on a board measuring 13.5 x 9 cms. fitted with three 10-way edge connectors, which allowed

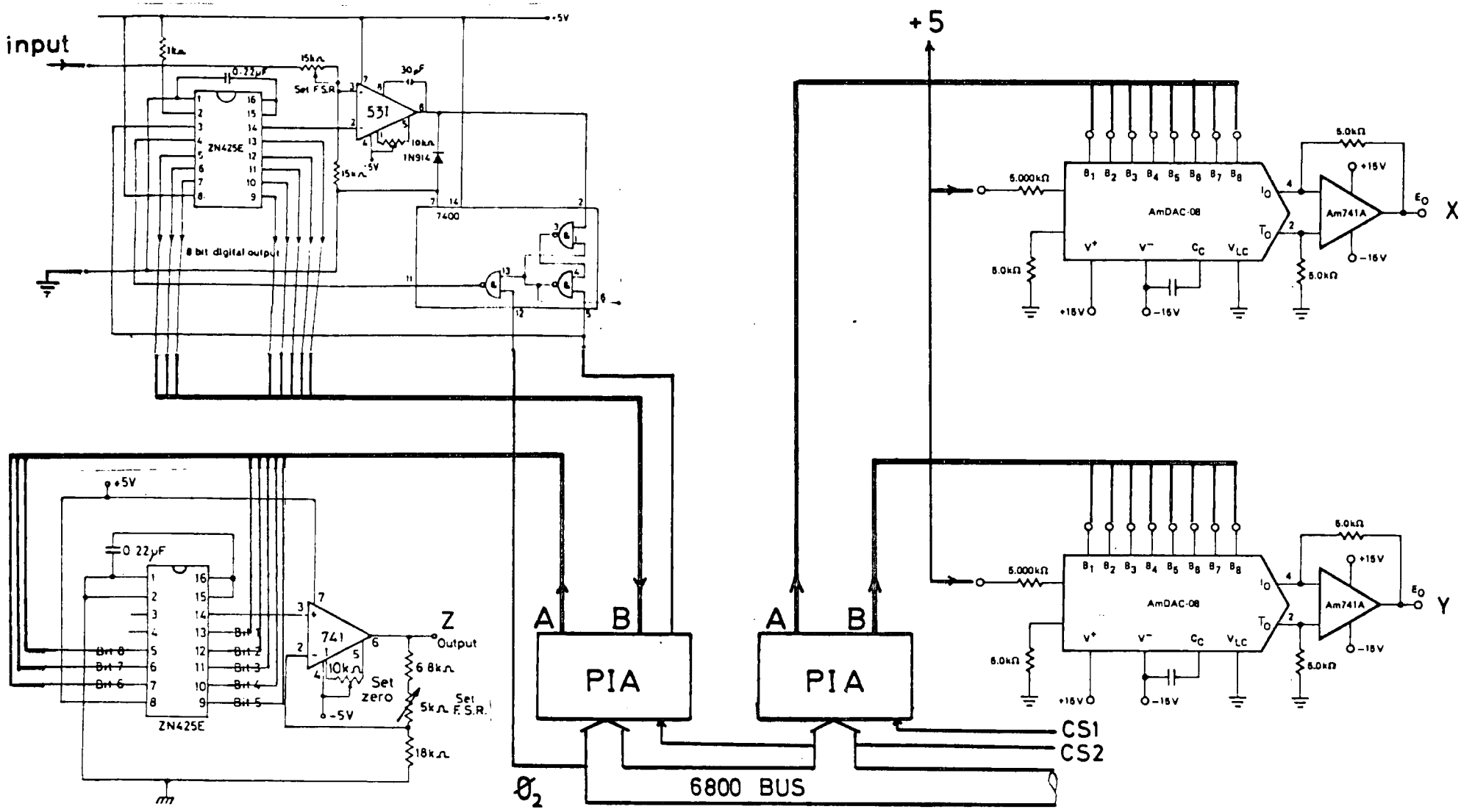


FIGURE 3.6. DIGITAL TO ANALOG / ANALOG TO DIGITAL CONVERSION SYSTEMS

the board to be plugged directly onto the SS-30 I/O bus of the microprocessor system. Printed circuit board techniques were used to define soldering pads for the IC's and all interconnections were made using wirewrap. The A to D converter was tested for linearity and conversion efficiency by plotting digital output against analogue input voltage at a 6kHz sampling rate using software timing. Linearity was found to be better than 0.1%, and a similar figure was obtained from the D to A system, plotting analogue output voltage against digital input.

3.3.3 The digital to analogue converters

Three D to A converters were required for the system; one to provide each of the analogue voltages required to define the coordinates for the display. The system used to define the z-modulation has already been described; two more converters were required for the x and y axes.

The two converters were constructed on the same board and were based around the AMD DAC-08 8-bit D to A converter IC. This component was chosen on grounds of economy (~£3 per chip) and has the added advantage of having an faster settling time (80ns) than many other converters of higher cost. The DAC-08 is a current output converter and requires external resistances to define an analogue output voltage. The internal reference amplifier requires a reference current which can be derived from a stabilised voltage source using an external resistor.

For this application the reference currents were derived from the +5V regulated supply using two 5kΩ resistors. This provided output currents in the range $-1mA \leq I_o \leq 0$. Inverting op-amp

circuits using 5kΩ feedback resistors were used to convert the output currents into voltages in the range $0 \leq V_o \leq 5V$ which were used as the output signals for the x and y axes.

3.3.4 Software

All software for the spectrogram system was composed using the microprocessor assembly language. Assembly-level programming is preferable for real-time applications where speed is of considerable importance. Efficient memory utilisation is possible, which cannot be acheived using a high-level language compiler. Programs were edited and assembled using the microprocessor development system together with the disc-resident editor and mnemonic assembler packages. The software can be divided into 4 parts:

(1) sampling and windowing

(2) computation of the Discrete Fourier Transform (DFT)

(3) power spectrum estimation

(4) plotting of results

Each of these will now be discussed in turn:

In order to estimate a power spectrum in the range $0 \leq f < 1/T$ Hz, it is necessary (by Nyquist's sampling theorem) to sample a signal at a rate of 2/T Hz. In this application the frequency band of interest was in the range $0 \leq f < 3$ kHz, necessitating a sampling rate of 6kHz. The sampling routine employed a software timing loop to define the interval between the acquistion of

samples from the A-D converter. This routine was used to acquire N 8-bit samples for each vertical line on the plot which were collected over a period of $N/(6 \times 10^3)$ seconds ($8 \leq N \leq 256$, $\log_2 N$ = integer). Each sample was converted to its 2's complement representation , scaled down by a factor N, and stored as the most significant byte in a dual byte storage location.

Where a finite length sequence of N samples is used to represent an infinite sequence, the finite sequence is the result of multiplying the infinite sequence by a rectangular 'window' sequence consisting of N samples of unity magnitude. The Fourier Transform of the resulting sequence is then the convolution of the transform of the infinite sequence with the transform of the rectangular window. The latter is of sin(x)/x form and produces undesirable side lobes in the power spectrum. These side lobes can be reduced by using a window which has unity magnitude at its centre but tapers to zero at each end. One such window is the 'Hamming' window, which was chosen for this application. For the rectangular window the first side lobe is only 13dB down from the main peak, whereas it is 40dB for the Hamming window (see figure 3.7). This extra suppression of the side lobes is acheived at the expense of a slightly wider main lobe.

The Hamming window is defined by the following equation:

$$w(n) = 0.54 - 0.46 \cos\left[\frac{2\pi n}{N-1}\right], \quad 0 \leq n \leq N-1$$

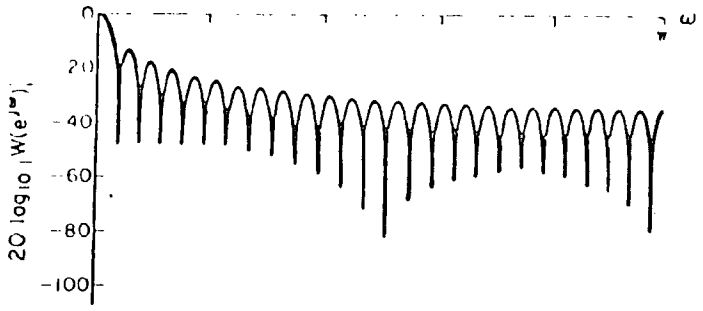


Figure 3.7(a). Fourier Transform of Rectangular Window.

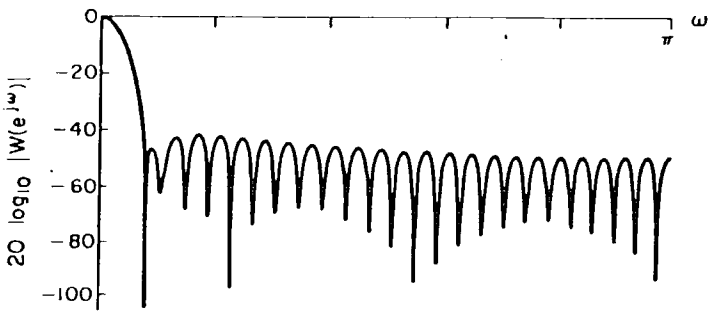


Figure 3.7(b). Fourier Transform of Hamming Window.

It is sometimes referred to as a 'raised-cosine' window. Multiplication of the input sequence by the window was achieved by storing samples of this function in a table in memory and multiplying each of the input samples by the corresponding stored window sample.

Evaluation of the DFT of the windowed input sequence was implemented using the Cooley-Tukey Fast Fourier Transform algorithm. This algorithm requires only $(N/2)\log_2 N$ complex multiplications per transform, as opposed to $4N^2$ multiplications for a direct implementation of the DFT equation. A detailed description of this algorithm and its implementation has been given in chapter 2. The DFT of a sequence $\{x(n)\}$ of N samples is given by:

$$X(k) = \sum_{n=0}^{N-1} x(n)\, e^{-j2\pi kn/N}, \quad k = 0,1, \ldots ,N-1$$

The resulting complex sequence $\{X(k)\}$ was stored in a table of 4N bytes; two bytes were allocated for each of the real and imaginary components of the X(k).

The power spectrum of the sequence $\{x(n)\}$ is defined as $\{|X(k)|\}$. The modulus of each of the X(k) was calculated by finding:

$$|X(k)| = \sqrt{X(k)_{RE}^2 + X(k)_{IM}^2}$$

The squares were computed using a 16 by 16 bit multiplication routine (using Booth's algorithm (16)) and the square roots were estimated using the Newton-Raphson recursion formula.

3.4 Results

Vertical calibration of the spectrogram was checked by applying sine waves of different frequencies to the system input and noting the vertical positions of the horizontal lines produced on the display. Each of the eight 1cm divisions on the y-axis represented 375Hz on a linear frequency scale. Gains of the x and y display ampifiers were adjusted to allow the display to fill the storage oscilloscope screen. A DC level produced a horizontal line across the bottom of the screen; a 3kHz sinusoidal input produced a horizontal line across the top. Figure 3.8 shows the display obtained with a 1.5kHz, 5v pk-pk sinusoid with a 3v superimposed DC level.

The audio output from a Racal RA17 HF receiver was filtered and used as the input to the spectrogram system. Unfortunately the receiver was found to have a slightly unstable BFO, which tended to spread the signal in the frequency domain. For all of the following results time is quantised along the x axis into 2^7 points, and frequency is quantised along the y axis into 2^6 points. This produces a display of 8192 discrete points in time-frequency space. The delay loop parameter was adjusted to produce a complete frame in 15 seconds.

Figure 3.9 shows the display obtained while monitoring a high-speed (~40 wpm) morse code transmitter centred on approximately 4.7MHz. Some intermittent narrow-band noise can be seen at the high frequency end of the channel (ie. at the very top of the display). Fades of up to one second were observed which are indicated by gaps in the displayed signal. Occasional

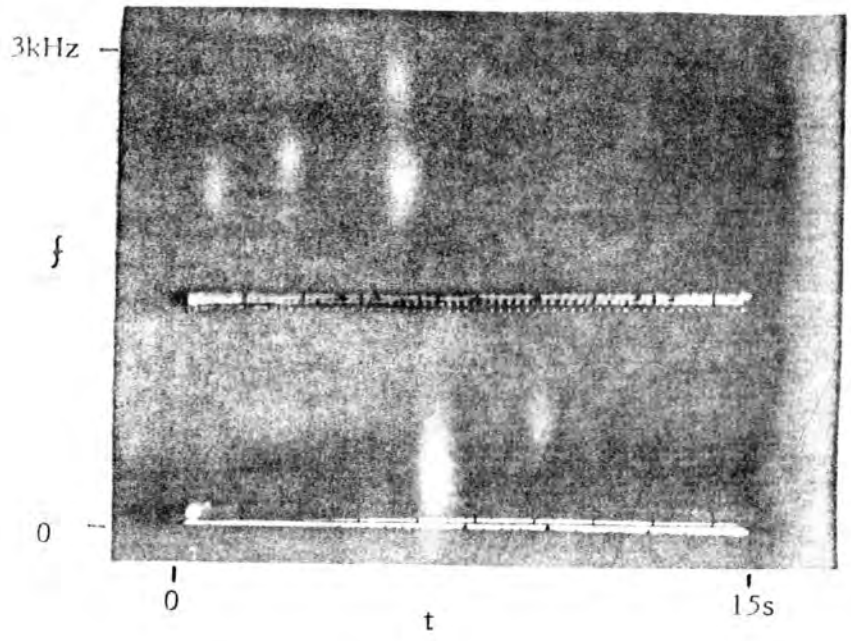


FIGURE 3.8

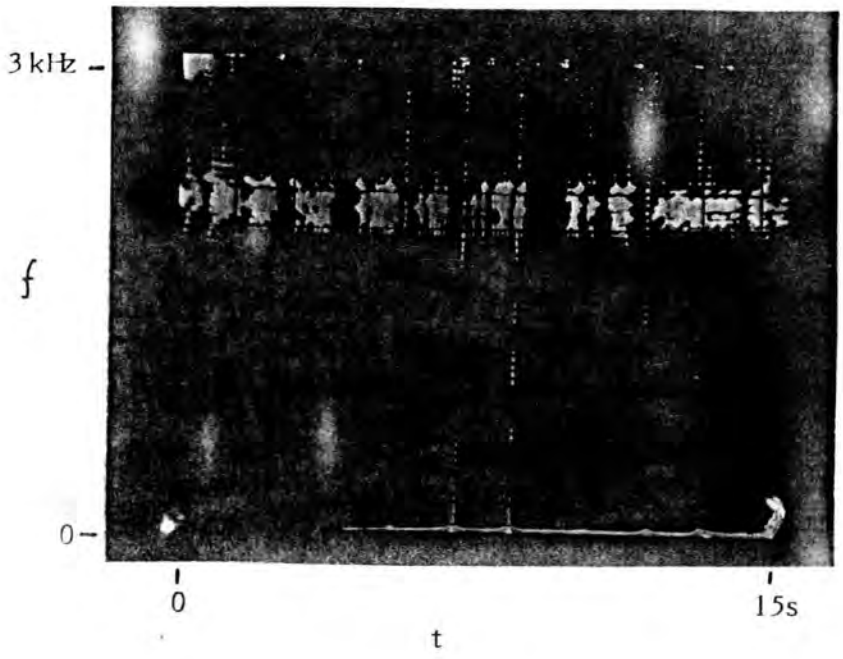


FIGURE 3.9

wide-band noise bursts are displayed as vertical lines traversing the whole channel.

Figure 3.10 is the display obtained while monitoring a voice channel containing two high-speed (~40 wpm) CW transmitters both of which were of equal average strength and separated in frequency by approximately 1kHz. It was found difficult to audibly decode either signal using a 3kHz receiver bandwidth. The two signals can be clearly seen on the spectrogram display. Frequency selective fades were observed audibly and can be seen on the display; ie. the horizontal positions of a gap in one signal (indicating a fadeout) do not always correspond to gaps in the other. A few broad-band noise spikes are again evident as thin vertical lines across the display.

The next two displays were the result of monitoring two 2-tone FSK signals of different data rates. The first (figure 3.11) shows the display resulting from observation of an estimated 150bps signal. The two tones can be clearly seen; short fast fades and bursts of noise are visible and were confirmed by audible monitoring of the receiver output. The display of figure 3.12 illustrates a 2-tone FSK signal of a higher data rate, estimated at 300bps. The frequency spreading is greater than for the slower-rate signal, although in both cases the spread was larger than expected, mainly due to the BFO instability. Frequency selective fading is evident from the virtual disappearance (for the first 4 seconds of the display) of the higher frequency tone. Broad-band noise was more severe than for the previous signal.

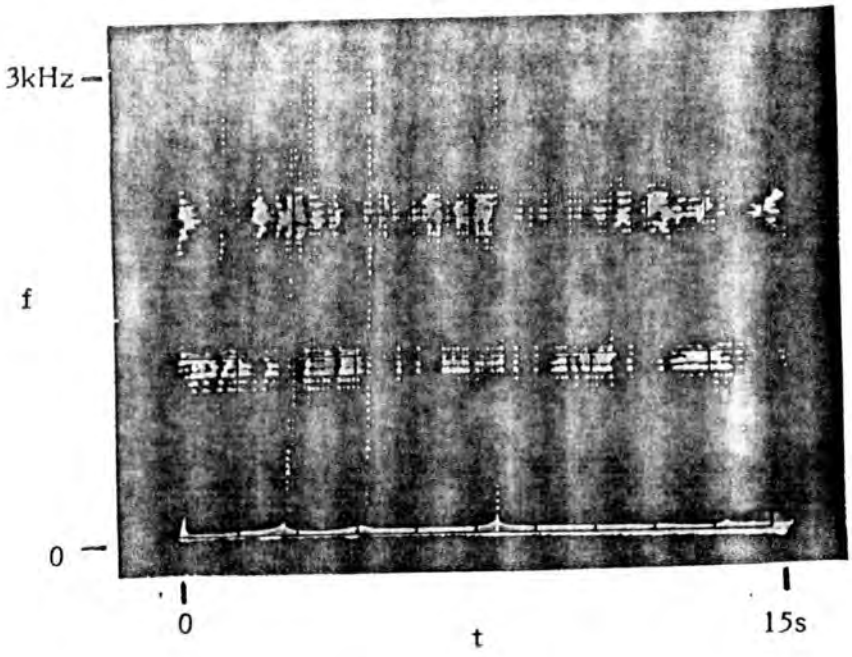


FIGURE 3.10

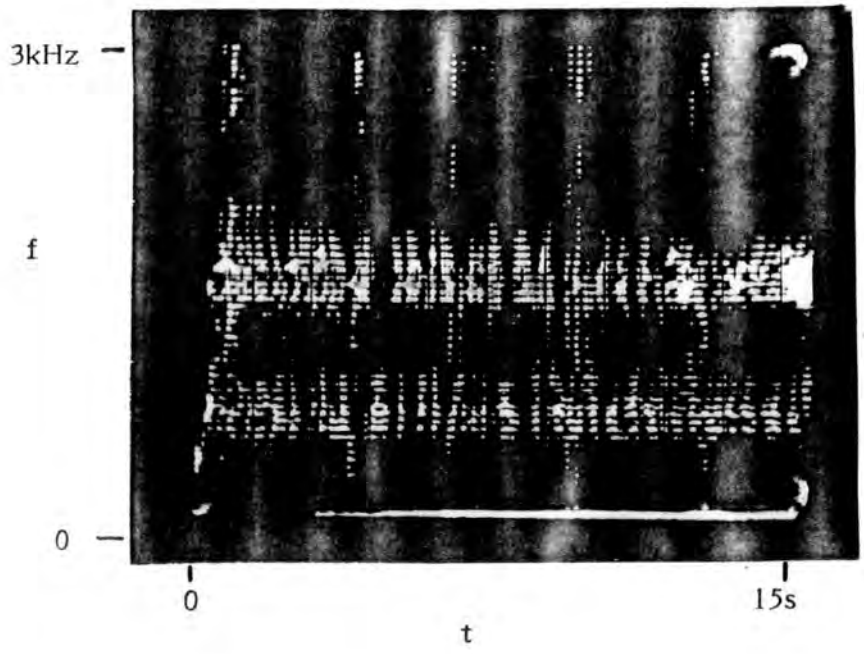


FIGURE 3.11

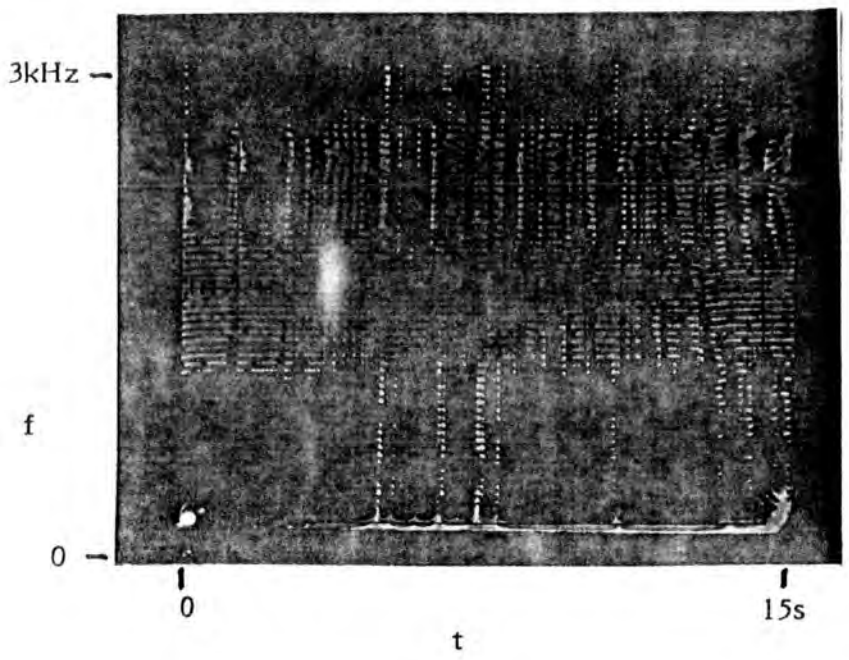


FIGURE 3.12

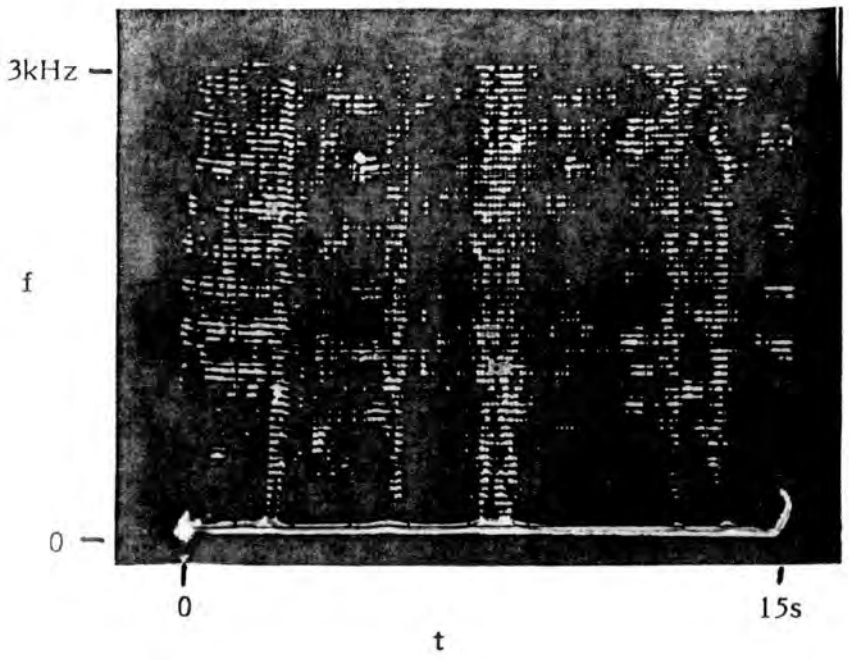


FIGURE 3.13

Figure 3.13 shows the spectrogram obtained from an AM broadcasting station centred on 7.42MHz observed during the early evening. The station was broadcasting orchestral music and was therefore broad band. Very deep frequency-flat fades were observed which can be identified by the large vacant areas in the display.

3.5 Conclusion

This chapter has described an economical microprocessor-based system for evaluation of an HF radio voice channel. The pictorial representation of the time-varying spectral properties of the channel enables the predominant disturbances to be identified. Broad-band noise bursts and narrow-band interference may be observed and fades on a known signal may be identified as frequency-flat or frequency-selective. The microprocessor implementation allows the system to present information in real time, which an operator may use to assess the suitability of a radio channel for data transmission.

Use of the spectrogram over a period of weeks indicated that the predominant disturbances tended to be narrow-band interference from other users of the channel. In most cases the spectral occupancy was limited to less than 20% of the overall bandwidth; this is examined in more detail in chapter 6. The results from the spectrogram would tend to indicate that benefit may be obtained from either (a) dynamic channel selection, where a change of channel frequency is made, or (b) dynamic in-band frequency allocation, where the spectrum of the transmitted signal is arranged to occupy the interference-free regions of the channel.

Frequency-selective and frequency-flat fades were observed from spectrograms of known signals. Frequency selective fades were observed to traverse the channel completely, usually in a short time (< 2s). Broad band noise bursts encompassing the whole of the channel spectrum were frequently observed. A data communications system for use over HF channels should therefore provide protection against long-term narrow-band interference phenomena and short-term broad-band fading and noise.

CHAPTER 4 The Slave Processor System

4.1 Introduction

It has been mentioned in the introductory chapter that the data processing tasks for some applications described in this thesis exceeded the capability of a single microprocessor unit. An example is the parallel HF modem transmitter, to be discussed in chapter 7, in which a modulated multitone waveform is to be generated digitally while simultaneously encoding and interleaving incoming data. The modem receiver is required to demodulate the received signal waveform and to decode the demodulated data. The serial processing capability of a single microprocessor is not sufficient to complete the required programming tasks in the available time.

As a result of these requirements, a small, self-contained microcomputer unit was developed which could perform a proportion of the processing tasks required by the overall system. This unit is connected into the system in a "master-slave" configuration such that the central (or "master") processor can assign tasks to one or more local (or "slave") processors. The slave processors then operate transparently to the central processor freeing the latter to perform other system tasks, returning at a later stage to restart or reallocate tasks as necessary. This form of distributed processing is useful where many of the system operations are repetitive, and can be implemented in microprocessor systems at low cost and with a considerable increase in overall processing power. This chapter describes a system implemented using the Motorola 6800

microprocessor which has subsequently been used in the HF transmitting and receiving equipment to be described in later chapters.

The task to be performed by the slave unit is loaded by the master into a localised 1 kbyte of RAM to which only the master and the slave concerned have access. Parameters to be processed by the task are also transferred into the slave processor's RAM and the task is initiated by a reset sequence on the slave processor which is under control of the master. Several slaves may be initialised by the master in this way, which are later checked using status bytes located within the RAM area to ascertain that the allocated tasks have been successfully completed. The results of a processed task are extracted from the slave memory and the task restarted using a different set of parameters, or a new task may be allocated. Because of the way in which the hardware is configured it is also possible for the master to dynamically access the slave memory without disturbing the flow of execution of the slave program. This can be useful when the slave is performing a task which is to be executed continuously, such as the generation of a voltage waveform in real time.

As an example of an application for such a system, suppose it is required to generate a sinusoidal waveform of which the frequency, amplitude and phase may be varied. The frequency parameter may specify the length of the steps to be taken through the lookup table so that if every sample is selected the frequency is f, if every 2nd sample is taken (step length = 2) it

is 2f and so on. The samples may be multiplied by a factor, A, to determine the amplitude, and the phase can be modified by specifying the starting point in the table. The selected sample values are converted into a real waveform by addressing a parallel interface connected between the slave processor unit and a digital-to-analogue converter. This is one example of many applications for such a system. Some of the system requirements and principles of operation are now discussed, followed by a description of the implementation.

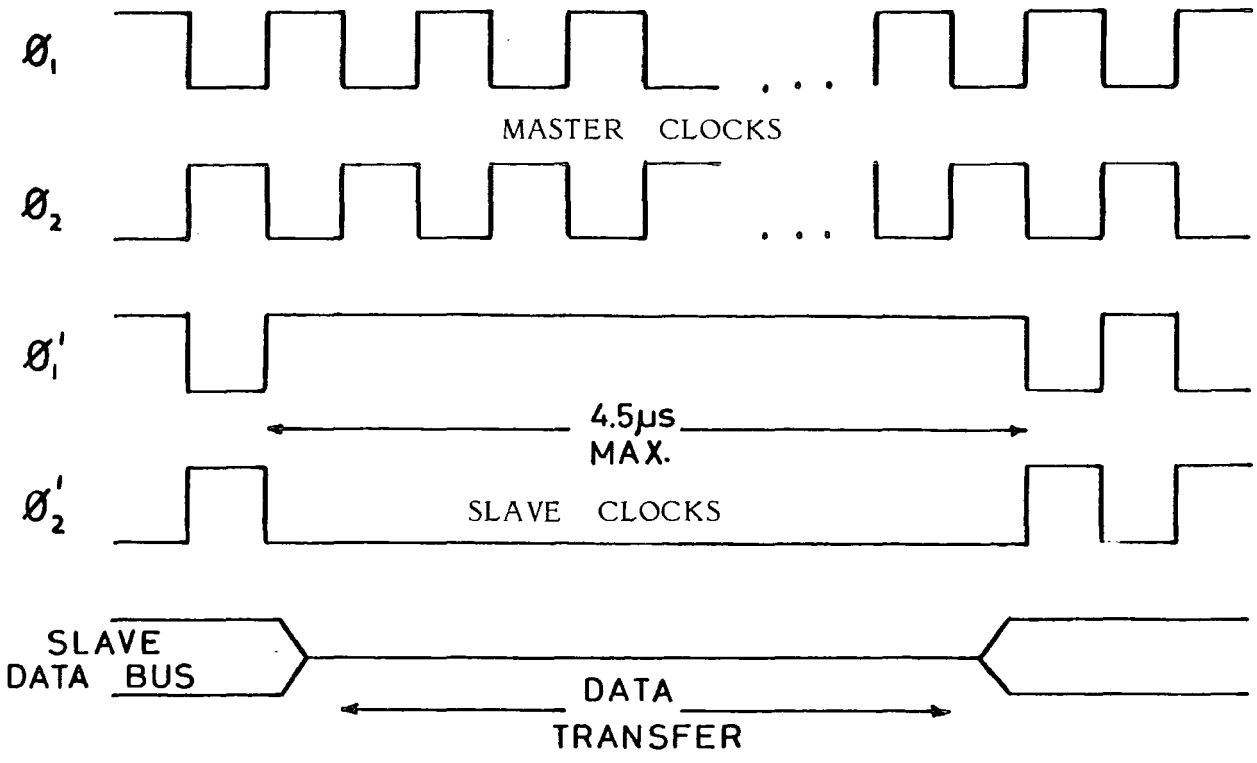
4.2 Principles of operation

The master-slave interface requires that the slave processor unit appears to the master as a continuous area of memory which may be read from, or written to, by the master processor regardless of the operation of the slave. Another requirement is that the master may have access to the more important slave processor control lines. It was decided that this may be most easily accomplished by assigning the lowest address of the slave memory area to a write-only control latch which is available only to the master unit. In this way the reset and interrupt sequences may be controlled by the master.

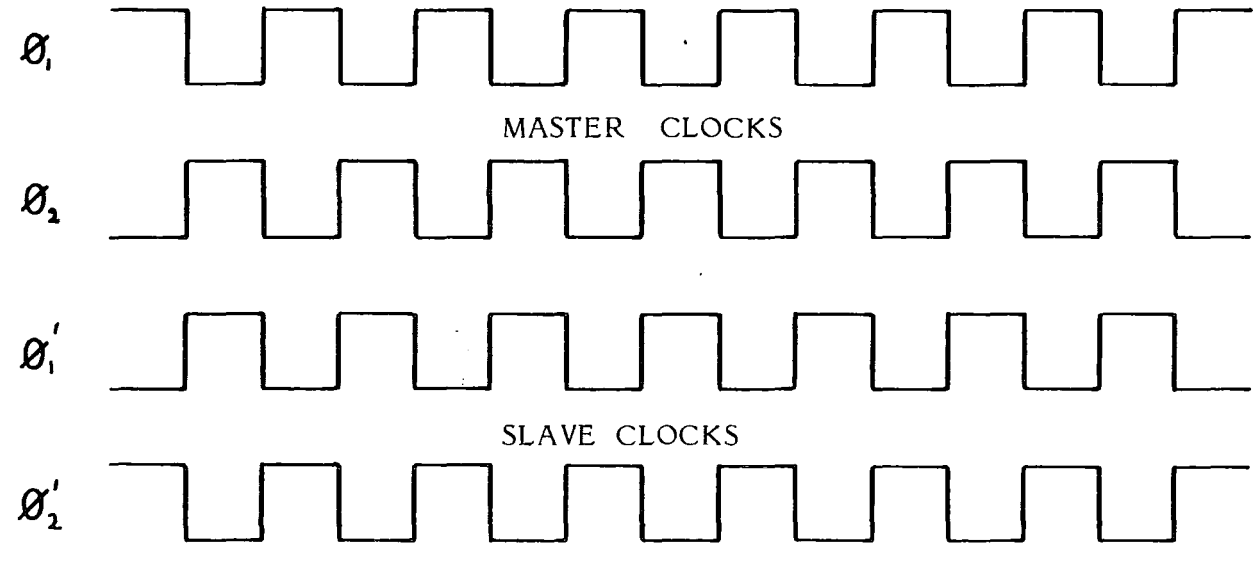
The requirement that the master and slave processors may attempt to simultaneously access a common area of memory may lead to conflicts when both are attempting to access the same byte. Possible ways of resolving address conflicts may be found by examining the requirements for the 6800 microprocessor clocks: A biphase clock of frequency not greater than 2.0MHz must be provided in which the two phases are non-overlapping. The phases

are designated $\emptyset_1$ and $\emptyset_2$ and are used to synchronise all data transfers to and from the microprocessor. The processor sets up an address during $\emptyset_1$ which becomes stable during the first half of $\emptyset_1$ and is stable throughout $\emptyset_2$. Data transfer (in a direction determined by the state of the read-write line) occurs during the fall of $\emptyset_2$ when the byte of data on the data bus is latched into the microprocessor or into memory. In the normal system configuration recommended by Motorola (38), the memory is allowed at least half of $\emptyset_1$ and all of $\emptyset_2$ in which to respond. Two possible methods of arranging the clock signals to avoid addressing conflicts in the master-slave system are shown in figure 4.1 and are described in the following two paragraphs.

If a conflict is to occur, it will begin during $\emptyset_1$, when an address is set up which is inside the slave memory area. This may be avoided by suspending execution of the slave processor program until access by the master is complete. This can be achieved by holding, or "stretching" the clock line to the slave processor while at the same time removing the slave from the busses by multiplexing the address lines and placing the data lines in high-impedance (tri-state) mode. However, because the internal registers of the 6800 CPU are dynamic, the clock may be stretched only to an upper limit of 4.5μs, beyond which destruction of internally held data may occur (39). In the situation where the slave RAM is being continually accessed by the master processor this limit may inadvertently be exceeded. It is for this reason that the following alternative method for avoiding conflicts was adopted in the final system.



(a) Clock Stretching



(b) Antiphase Operation

FIGURE 4.1 ADDRESS CONFLICT AVOIDANCE

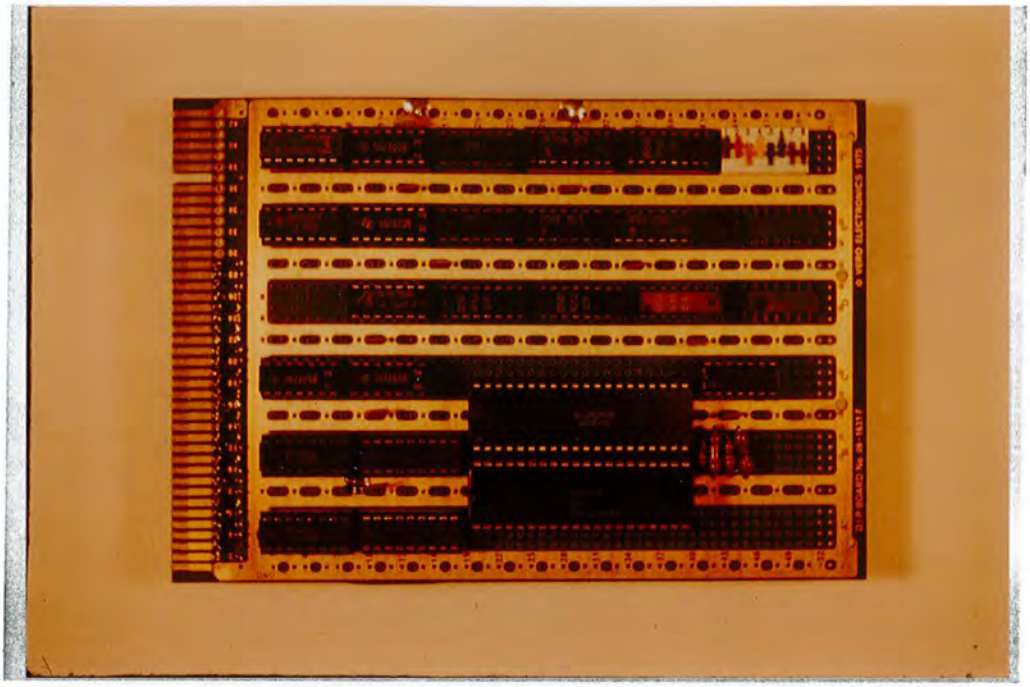


PHOTO 4.1(a). SLAVE PROCESSOR PROTOTYPE BOARD (TOP).

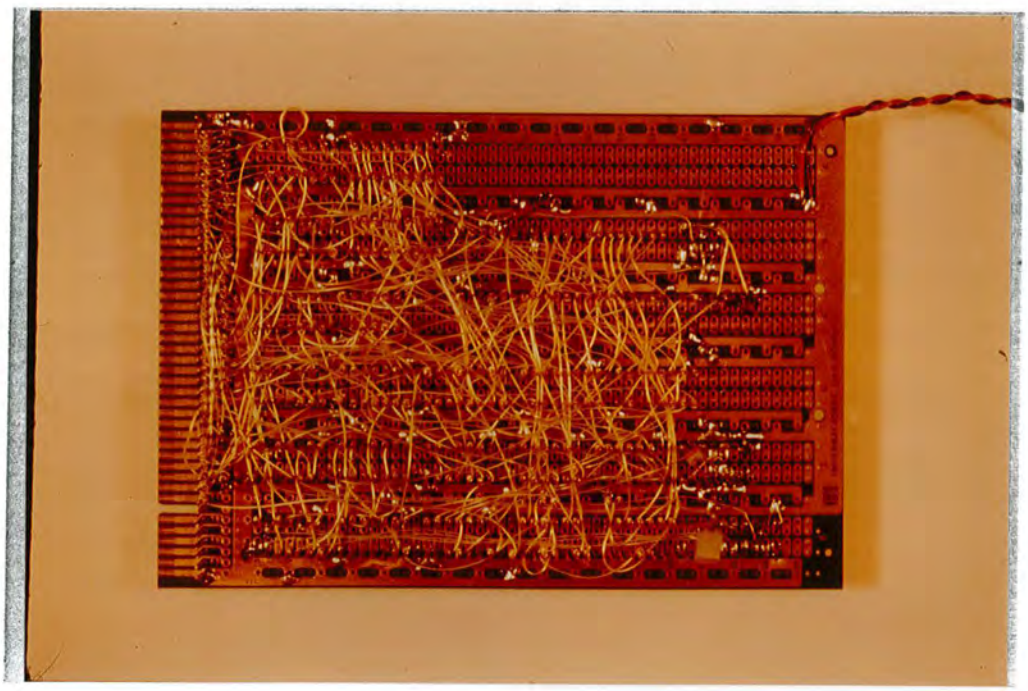


PHOTO 4.1(b). SLAVE PROCESSOR PROTOTYPE BOARD (UNDERSIDE).

By running the two processors in antiphase, the slave processor is in $\emptyset_2$ when the master is in $\emptyset_1$ and vice versa. Furthermore, the address bus is multiplexed so that the address lines corresponding to whichever processor is in $\emptyset_2$ are always connected to the slave RAM. The master may access the RAM provided the block is selected by decoding the high order address lines of the master processor address bus. Memory access is permitted only during $\emptyset_2$ of either processor, which places an upper limit on the memory access time of one half of the clock period, since only $\emptyset_2$ is available for memory address set-up. Nevertheless, static RAM with a sufficiently short access time is available at reasonable prices and this latter method for avoiding address conflicts was chosen in preference to clock stretching.

This completes the description of theoperating principles of the master-slave configuration, the constituent components of which are now discussed in more detail.

4.3 Implementation

Implementation of the slave processor system is outlined in the block diagram of figure 4.2 in which the various functions are grouped into a set of distinct units:

(1) block select logic

(2) address line multiplexers

(3) data tri-state buffers and enable logic

(4) control latch and enable logic

(5) clock drivers

(6) memory and enable logic

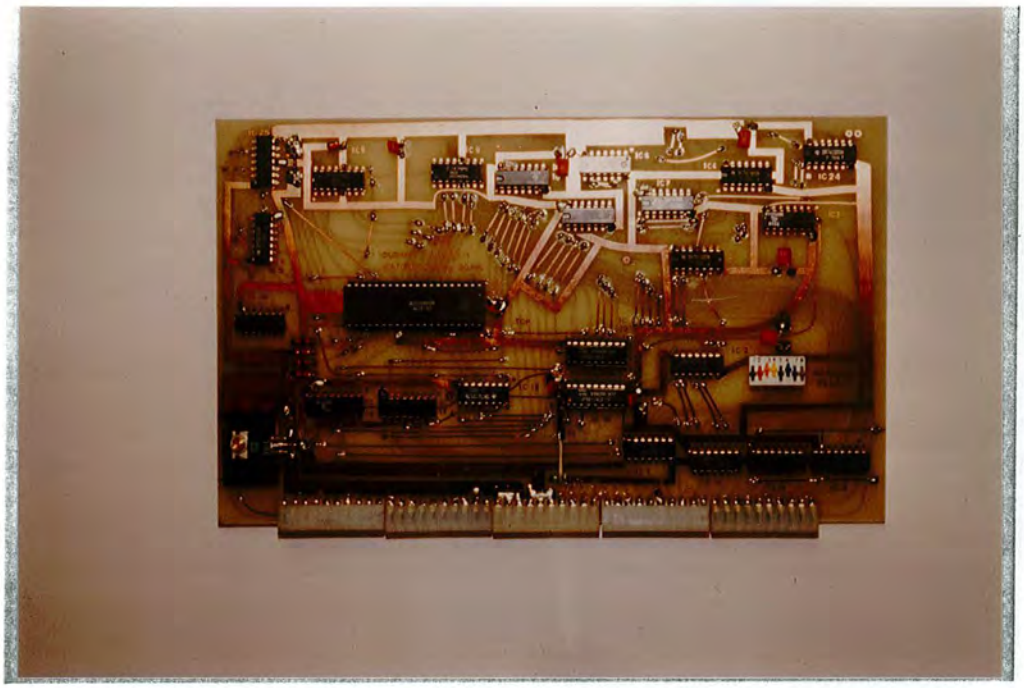


PHOTO 4.2. SLAVE PROCESSOR UNIT. PRINTED CIRCUIT BOARD IMPLEMENTATION

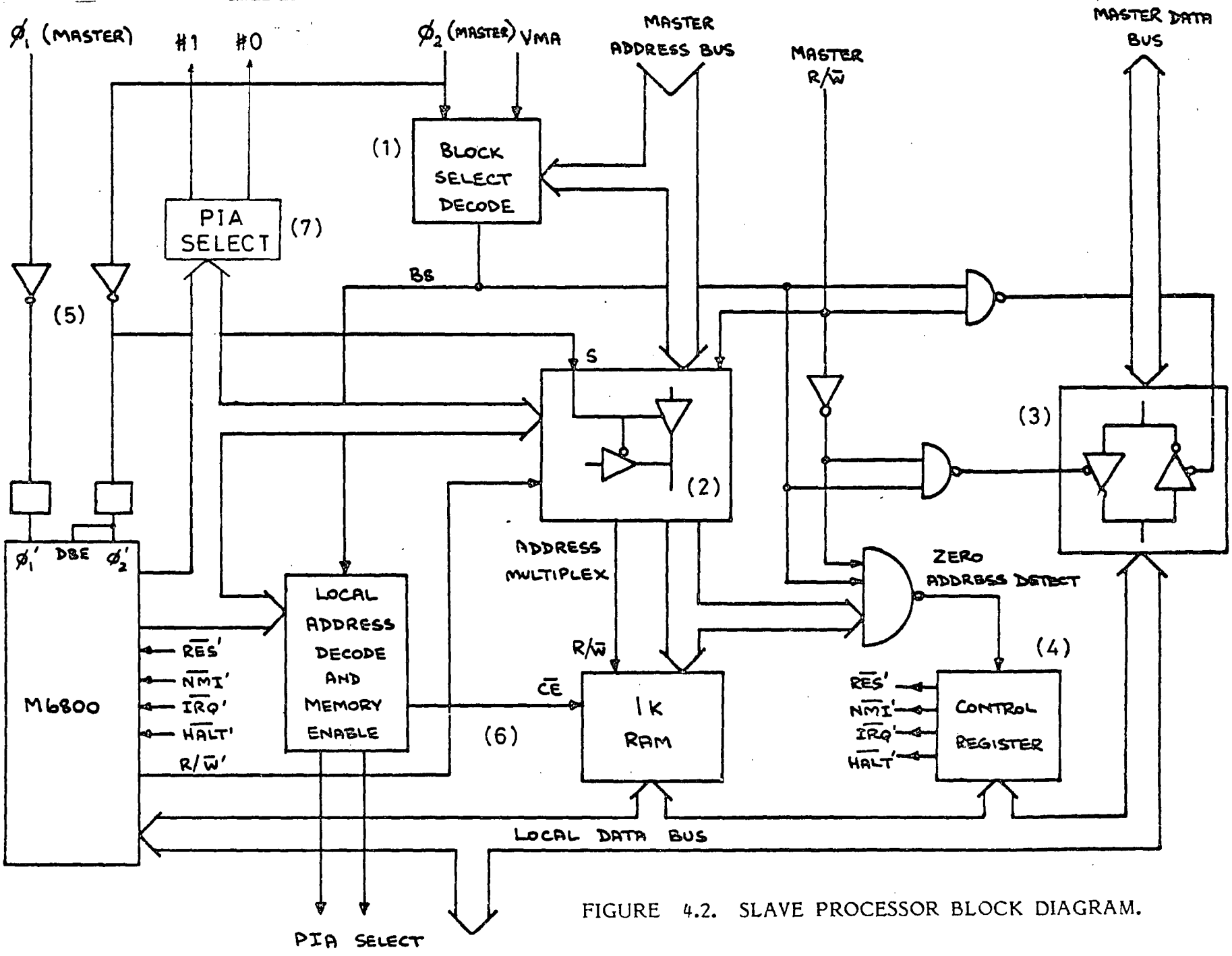


FIGURE 4.2. SLAVE PROCESSOR BLOCK DIAGRAM.

(7) PIA enable logic

The functions and implementation of each unit are discussed separately and reference should be made to the circuit diagram of figure 4.3 ((a) and (b)).

4.3.1 Block select logic

Each of the two inputs of six exclusive-OR gates are connected to a switch and to one of the six high-order address lines respectively. The other sides of the switches are grounded and the gate outputs are combined using two NOR gates and a single NAND gate. The switches may be used to manually locate the 1kbyte of memory anywhere on a 1k boundary within the available address space. VMA and $\overline{\phi}_2$ are also included in the decoding to ensure that the address received is a valid one. In a system which uses more than one slave processor, each may be switch selected to reside within a different segment of memory.

4.3.2 Address multiplexers

The ten low-order address lines from the master processor and the R/$\overline{W}$ line are multiplexed with the low-order slave processor address lines into the local memory address bus. The local ϕ_2 clock is used to control the address routing.

4.3.3 Data bus buffers

The data bus interface is provided by two bi-directional tri-state buffers. The two control lines, $\overline{\text{Transmit Enable}}$ ($\overline{\text{TE}}$) and $\overline{\text{Receive Enable}}$ ($\overline{\text{RE}}$), allow three possible functions: Data is passed from the master processor data bus to the slave data bus, or from the memory to the master data bus, or both sides of the

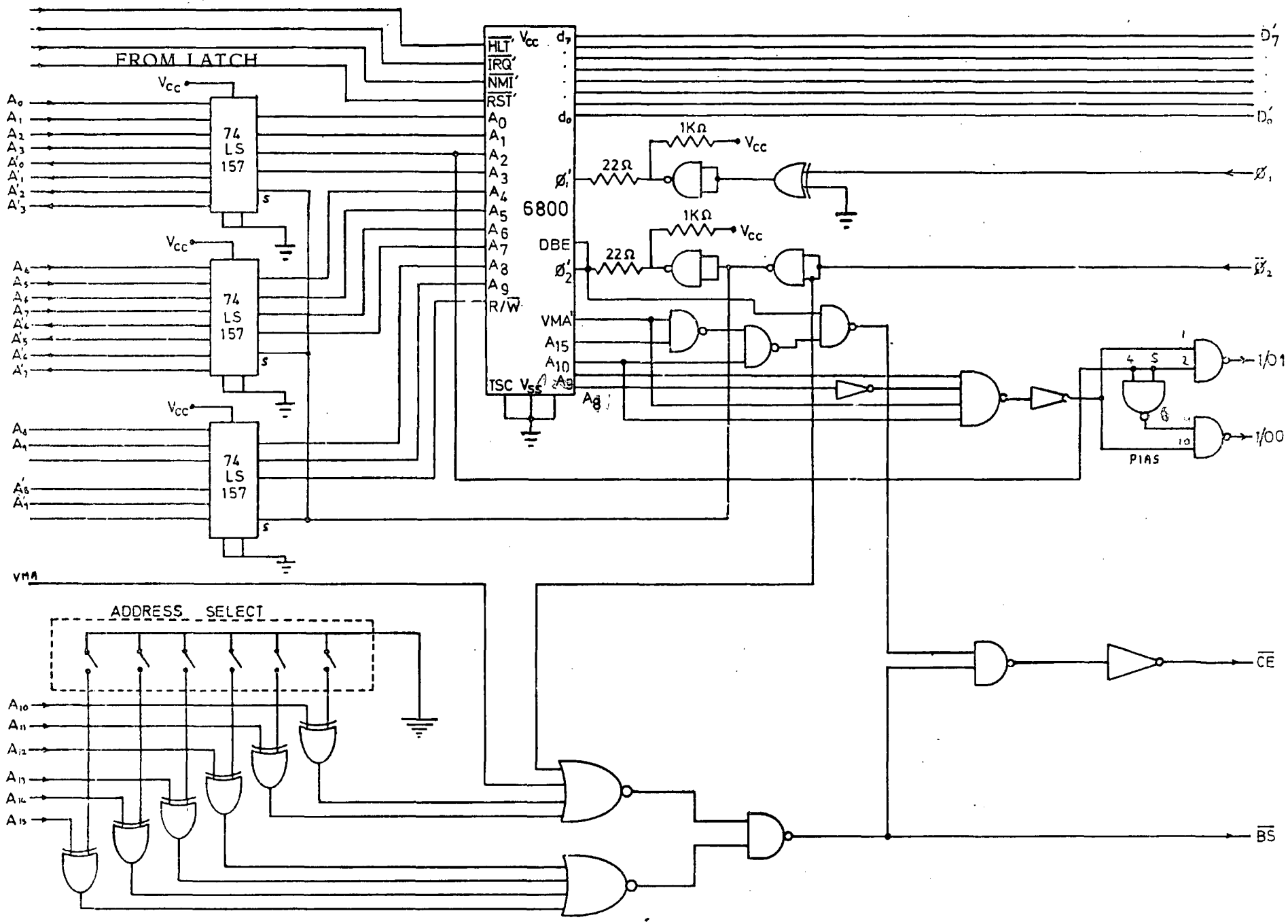


FIGURE 4.3(a). SLAVE PROCESSOR SYSTEM. (address multiplexers, block select, PIA select.)

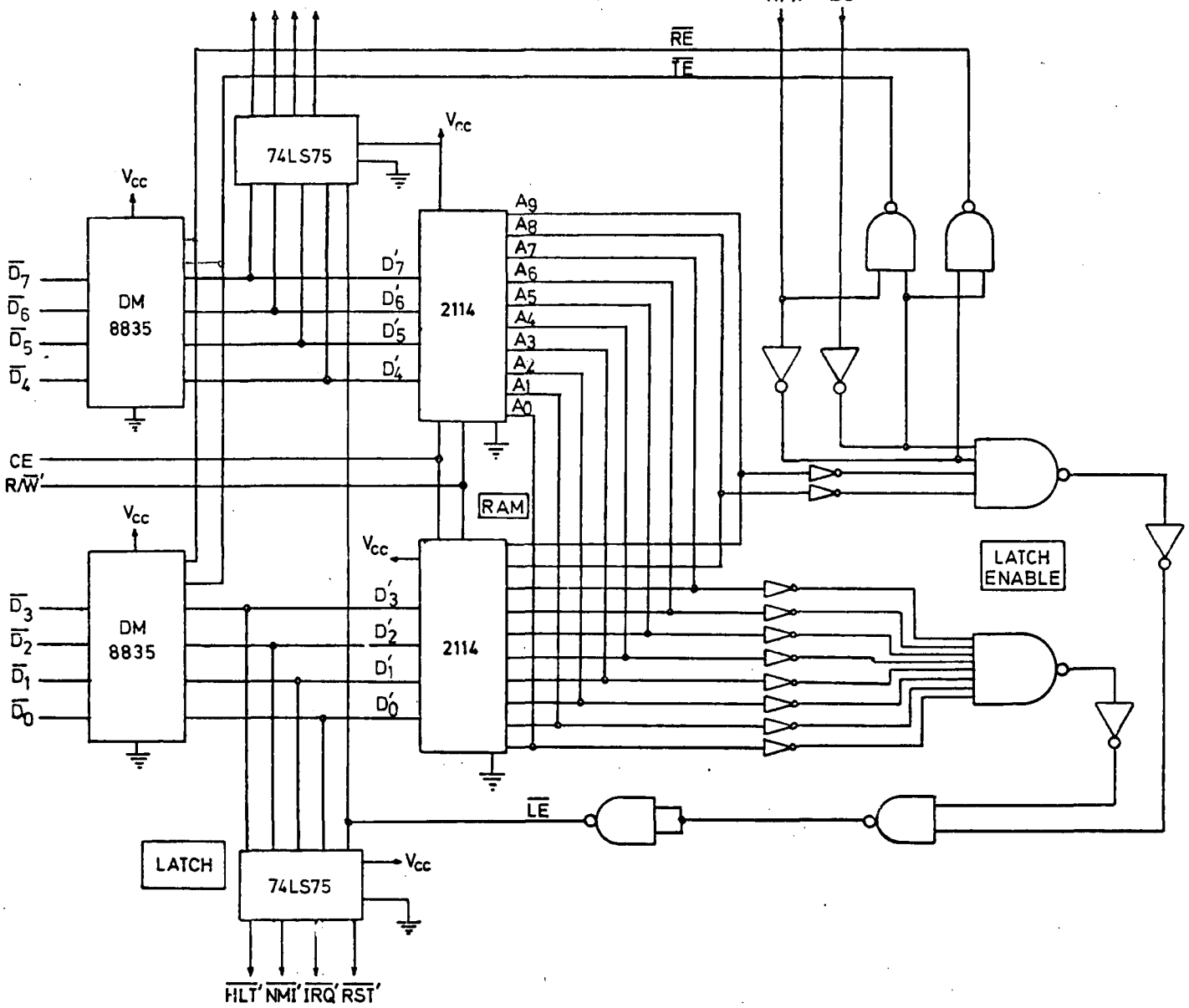


FIGURE 4.3(b). SLAVE PROCESSOR SYSTEM (Data buffers, memory, control latch).

buffers enter the high impedance state thereby preventing interaction between the two busses. This latter state is required to isolate the slave unit from the master when the master is accessing a memory location outside the boundaries of the slave address space. $\overline{RE}$ and $\overline{TE}$ are derived from NAND operations of BS with $\overline{R/\overline{W}}$ and $R/\overline{W}$ respectively and determine the direction of data flow (if any).

4.3.4 Control latch

Two quad latches are selected to reside at the base of the slave processor address space by using ten inverters on the low order address lines to detect the 'zero' address. BS has been included in the select logic to uniquely identify a particular pair of slave latches within the overall system and $R/\overline{W}$ is used to ensure that the latches are "write-only". The four 'Q' outputs from one latch are connected to the $\overline{RES}$, $\overline{NMI}$, $\overline{IRQ}$, and $\overline{HALT}$ control lines of the slave processor, and the outputs from the second latch are left unconnected, to be user defined at a later stage. Data may be written to, but not read from, the control latch by the master processor. Because the latch is invisible to the slave processor, the bottom memory location in the RAM may by freely used by the slave and may also be read by the master.

4.3.5 Clock drivers

The clock signals required for the 6800 microprocessor are not TTL compatible and need to be derived using open-collector drivers with pullup resistors. The antiphase operation of the slave unit (with respect to the master) requires that the clock

signals from the master bus be inverted. This is acheived using a NAND gate as an inverter and an exclusive-OR gate with one input grounded. The latter has been included to equalise gate delays which might otherwise cause unacceptable overlapping of the two clock phases.

4.3.6 Memory

Two RAM IC's, each having a capacity of 1k x 4 bits and an access time of 150 ns were used to provide 1k bytes of continuous memory capable of operating with clock frequencies in excess of 2.0 MHz. The enable lines were tied together and to the output of the chip enable circuitry. The memory must be enabled (a) when the master processor addresses the slave memory block, (b) when the slave processor addresses the lowest 1k bytes of its address space, and (c) when the slave processor addresses any of the eight top locations of its address space ($FFF8-$FFFF) which contain the interrupt and restart vector pointers. Two external chip enable lines are provided to enable two 6821 Peripheral Interface Adapters (PIA's) which must remain inactive during memory access cycles. The truth table of table 4.1 determines the select logic required for memory enable decoding. Implementation was acheived using four NAND gates and one inverter. VMA and $\emptyset_2$ were included to permit only valid addresses.

4.3.7 PIAs

The PIA select circuitry allows the inclusion of two 6821 chips in the slave processor system. Each PIA occupies 4 addresses and address lines A12, A11, A10 and A2 were decoded to

locate the PIAs at $0C00 to $0C03 and $0C04 to $0C07. Address lines A1 and A0 were used to reference the internal PIA registers. The truth table (table 4.1) determines (1) that the memory is disabled when selecting a PIA, and (2) that the PIAs may be addressed by the slave processor only.

A_{15}	A_{10}	A_9	A_8	$\overline{CE}$	$\overline{PIAS}$	
0	0	X	X	$\emptyset_2$	1	Internal RAM
0	1	0	0	1	1	
0	1	0	1	1	1	
0	1	1	0	1	1	
0	1	1	1	1	0	PIA select
1	X	X	X	$\emptyset_2$	1	Reset vector

Table 4.1

This completes the description of the functional blocks of the slave system. The prototype was constructed on a standard wirewrap board measuring 6.5 x 4.5 ins. and included a single PIA. Photos 4.1(a) and 4.1(b) show top and underside views of the assembled system. SSI components were chosen from the 74LS' series of IC's, and high frequency versions of the LSI chips were used for 2MHz. operation. Connection to the master system was via a 32-way ribbon cable approximately 0.5m in length. Initial tests on the system proved unsuccessful when operating at a clock frequency of 2.0 MHz; however a successful series of diagnostics were performed at 1.0 MHz which will now be described.

4.4 Test Results

The switches on the slave processor board were set to locate the RAM at addresses $C000-$C3FF. This is acheived by the following combination, where a '1' indicates a closed switch:

SW5	SW4	SW3	SW2	SW1	SW0
0	0	1	1	1	1

Since the latch is always located at the base address of the slave RAM, its global address in this case was $C000. The data format of the latch was defined by the system hardware to be:

d7	d6	d5	d4	d3	d2	d1	d0
X	X	X	X	$\overline{\text{HALT}}$	$\overline{\text{NMI}}$	$\overline{\text{IRQ}}$	$\overline{\text{RES}}$

The following diagnostics were performed to ensure that the slave system was functioning correctly. The assembler written programs referred to in the text were assembled using the co-resident mnemonic assembler.

(1) Memory diagnostic. The slave processor was halted by using the system monitor to write $00 to the slave latch. The memory diagnostic program 'CDAT-1', written by John Christenson of Motorola Inc. (Appendix 2), was used to test the slave memory (with the exception of the latch address) for faulty bits and convergent address problems. Successful execution of this test ensured that the memory was working satisfactorily.

(2) Reset/interrupt sequence testing. The program listed in Appendix 2 was used to test the slave reset and interrupt operations. For a base address of $C000, the reset and interrupt vector pointers reside at the following global addresses:

$C3F8-9	IRQ
$C3FA-B	SWI
$C3FC-D	NMI
$C3FE-F	RES

The local addresses of the interrupt and reset sequences were loaded into the corresponding vector locations and each sequence

was checked by toggling the appropriate interrupt/reset line to initiate the appropriate sequence. Toggling was done by using the system monitor to write to the slave latch. Each test sequence was designed to write a particular byte into global location $C001 (local address $0001). This number could be read by the system monitor to check for correct execution and could also be dynamically altered without affecting execution flow of the slave program.

(3) Parallel processing. The third diagnostic was used to demonstrate the parallel processing capability of the master-slave configuration. The example chosen was to evaluate the expression:

$$(a \times b) + (c \times d)$$

where a,b,c and d are 2-byte 2's complement numbers. The program listed in Appendix 2 uses two multiplication routines; one in master RAM, the other in slave RAM. The two numbers a and b were used as arguments for the slave; b x c was evaluated by the master. A comparison of the execution times required for evaluation of the expression by the master only, and by the master-slave configuration showed a two-fold increase in speed by the latter over the former, as would be expected. Note that in programs which require a slave system stack, as in this example, it is necessary to define a local stack pointer; initialising the pointer immediately below the vector space allows the stack to extend downwards through the slave memory.

4.5 Printed circuit construction

The slave circuit was transferred onto a dual sided printed circuit board designed to fit onto the SS-50 bus using standard

Molex connectors. The artwork was drawn twice full size using transfers and tapes, then photo-reduced to a correctly dimensioned "positive" image. The front and back images were carefully aligned and fastened together at two sides to enable the blank board to be slid in between before exposing to ultra-violet light and etching in the usual way. The printed circuit version of the system was found to perform satisfactorily with a 2.0 MHz. clock frequency and three such boards were produced from the original mask. Photo 4.2 shows the printed circuit board implementation.

4.6 Conclusion

The design and implementation of a distributed microprocessor system has been discussed, in which a number of "slave" microcomputer units are controlled by a "master" processor. The master processor designates tasks to the slave units, and instructs them to execute those tasks when required. Parameters may be passed to and from the slave memory in much the same way as parameters are transferred to and from subroutines. However, by operating the slave processors in antiphase to the master, the slave memory may be dynamically accessed by the master, without disturbing execution of the slave program. The increase in processing power which may be achieved using such a system has been demonstrated with the use of examples; this advantage will become further evident in later chapters.

CHAPTER 5 Error-control coding

5.1 Introduction

From a technical point of view, a generalised data communications system may be regarded as consisting of three basic blocks: the transmitter, the channel and the receiver. The transmitter has the task of assigning an analogue waveform to each possible sequence of digits received as input from the data source. This is the process of modulation. The analogue waveforms are propagated through the channel and are then interpreted individually at the receiver so that the output of the receiver detector is a sequence of digits representing best estimates of the transmitted data. The channel in this case is known as the "modulation channel".

The above generalised communications system may be viewed in its entirety as a strictly digital channel. In the binary case this channel accepts 0's and 1's at its input and usually reproduces them at its output. Occasionally, however, because of noise and other channel impairments, the output digits do not agree with the input digits and errors have occurred. Each message is associated with a sequence of bits to be passed through the digital channel. In order that they may be distinguished, it is desirable to associate with messages bit sequences which are as different as possible from one another. This may be achieved by adding redundant bits to each message sequence so that a message sequence of k bits is transmitted as a block of n bits, where $n > k$. The communications system may now be regarded as having the form of figure 5.1, where the "encoder"

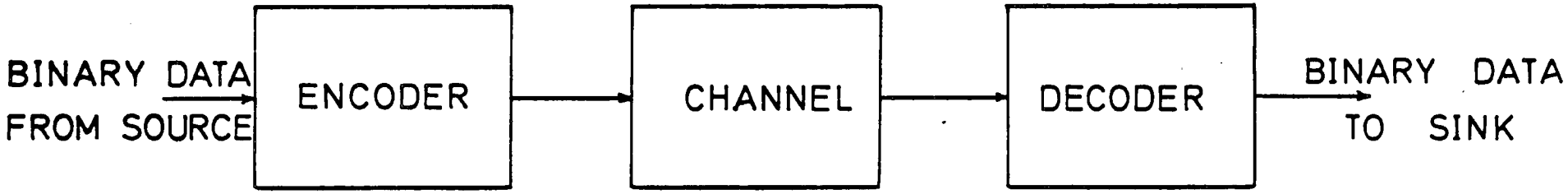


FIGURE 5.1 CODING CHANNEL MODEL

adds redundant bits to the source data in a systematic manner. The "decoder" removes the redundancy after transmission over the digital channel (known as the "coding channel") and may attempt to detect or correct errors introduced by the channel.

The components of the coding channel in HF radio systems are the HF transmitter, the HF radio path and the HF receiver. The transmitter and the receiver include the digital modulator and demodulator respectively. As a first step towards minimising the errors, close attention should be paid to the modulation scheme to reduce the effects of intersymbol interference and noise. However, the short-term variations in the characteristics of the HF channel are largely unpredictable, and the channel impairments often result in extremely high error rates. It is therefore desirable to add redundancy in the manner described if these errors are to be eliminated.

This chapter discusses the microprocessor implementation of block coding schemes for random error correction, and shows how the blocks may be interleaved to correct bursts of errors, such as those observed on the HF coding channel. Field test results are discussed in a later chapter.

5.2 Block codes

If the redundancy added to the message digits is to be utilised by the decoder for error control, the redundant bits must be added in a systematic and predetermined manner. An effective way is to use a parity check block coding scheme in which a number (n-k) of modulo-2 sums of (or parity checks on) various digits of a k-bit message digit sequence are computed and

appended to the information digits. The n-bit block is then shifted out onto the channel. The data rate is reduced by a factor k/n, known as the "code efficiency". After transmission through the coding channel, the same parity checks are computed at the decoder; if they do not agree, then errors must have been introduced by the channel. In the binary case, if it is possible to locate the errors, they may be corrected.

An (n,k) block code is defined as the collection of 2^k n-tuples produced by encoding all possible k-tuples according to some pre-determined set of parity-check rules. The encoding of a data block into a code word can be represented mathematically as:

$$\mathbf{c} = \mathbf{dG}$$

where $\mathbf{c}$ is an n-bit code word represented as an n-place row vector (n-tuple), and $\mathbf{d}$ is a k-bit data block represented as a k-place row vector (k-tuple). The k-by-n matrix $\mathbf{G}$ is the generator matrix of the code and has the form:

$$\mathbf{G} = [\mathbf{I}_k \mathbf{P}]$$

where $\mathbf{I}_k$ is the identity matrix of order k, and $\mathbf{P}$ is an arbitary k-by-(n-k) matrix. The leftmost k symbols of $\mathbf{c}$ are therefore identical to the corresponding symbols of $\mathbf{d}$, while the rightmost n-k symbols are modulo-2 sums of, or parity checks on, various symbols of $\mathbf{d}$.

In order that a code may detect up to t errors per codeword, the minimum Hamming distance between words in the code must be at least t+1. If the code is to correct up to t errors per

codeword, the minimum Hamming distance between codewords must be increased to 2t+1. The problem is to choose the matrix **P** to maximise the minimum distance between codewords.

5.3 Cyclic codes

Much of the research in coding theory has been concentrated on a small subclass of block codes, the cyclic codes (40). These codes possess a fair amount of mathematical "structure", allowing codes to be designed having good error-correcting properties and which may be implemented with a minimum of hardware or software. A cyclic (n,k) code, a linear block code of length n having k information symbols, has the property that every cyclic shift of a code word is another code word.

That is if:

$$\mathbf{c} = \{c_{n-1}, c_{n-2}, \ldots, c_0\}$$

is a code word, so are:

$$\{c_{n-2}, c_{n-3}, \ldots, c_{n-1}\}$$

$$\{c_{n-3}, c_{n-4}, \ldots, c_{n-2}\}$$

$$\vdots$$

$$\{c_0, c_{n-1}, \ldots, c_1\}$$

The elements of each code word can be treated as coefficients of a polynomial of degree n-1. The codeword can be represented as a code polynomial; that is,

$$c(x) = c_{n-1}x^{n-1} + c_{n-2}x^{n-2} + \ldots + c_1x + c_0$$

That c(x) is a code word implies that $x^i c(x)$ modulo-(x^n+1) is also a code word for all i. The polynomial representation of the

bottom row of the matrix **G** defined earlier is known as the "generator polynomial", g(x), of the code. All other rows in the matrix of a cyclic code are multiples of this polynomial; given the generator polynomial of the code, it is possible to construct the generator matrix (41). Since g(x) has degree n-k, there are 2^k polynomials, ie. those of degree less than k, which can be multiplied by g(x) to yield a polynomial of degree less than n. Clearly, there is a one-to-one corres,ondence between these polynomials and the 2^k words in the code. It may be proved (12) that the generator polynomial, g(x), is always a divisor of x^n+1 .

It may be shown (41) that encoding a k-bit data block by multiplying it by the generator matrix **G** is equivalent to the following polynomial operation. The polynomial representation of the information block, denoted by d(x), has degree less than k; therefore $x^{n-k}d(x)$ has degree less then n. Also

$$\frac{x^{n-k}d(x)}{g(x)} = q(x) + \frac{r(x)}{g(x)} \qquad (5.1)$$

where q(x) has degree less than k and r(x) has degree less than n-k, the degree of g(x). Thus the polynomial $c(x) = x^{n-k}d(x) + r(x)$ is divisible by g(x) and is a code word in the code generated by g(x). This word consists of the unaltered k-bit information block followed by n-k linear combinations of the

information bits.

If **c** is a code word in the code generated by the matrix G defined previously, then

$$\mathbf{c}\begin{bmatrix}\mathbf{P}\\ \mathbf{I}_{n-k}\end{bmatrix} = 0$$

Hence, any n-tuple **e** that is not a code word gives:

$$\mathbf{e}\begin{bmatrix}\mathbf{P}\\ \mathbf{I}_{n-k}\end{bmatrix} = \mathbf{s} \neq 0$$

The vector **s** is an (n-k)-tuple referred to as the "syndrome" of the n-tuple. Every n-tuple has one, and only one, syndrome. The syndrome is obtained by encoding the information section of an n-tuple and adding (modulo-2) the resultant check bits to the corresponding bits of the parity section of the n-tuple.

Let $e(x) = e_d(x) + e_p(x)$ where $e_d(x)$ and $e_p(x)$ are the data and parity sections of the n-tuple e(x) respectively. The syndrome s(x) is given by:

$$s(x) = e_p(x) + r(x)$$

where r(x) is the residue obtained by dividing $e_d(x)$ by g(x). But since $e_p(x)$ has degree less than g(x), this is identical to the residue obtained by dividing $e(x) = e_d(x) + e_p(x)$ by g(x). That is,

$$s(x) = \text{rem}\ \frac{e(x)}{g(x)}$$

Since g(x) divides every code word c(x), the syndrome is identical

to that of e(x) + c(x).

5.4 The BCH codes

The BCH (Bose-Chaudhuri-Hocquenghem) codes (42) are a class of cyclic codes of particular interest. They are defined in terms of the BCH bound, a statement of which is given later. The proof of this bound may be found in reference (12).

The "primitive" BCH codes are of length 2^m-1 (m integer) and require, at most, mt check bits to correct up to t errors per codeword. It has been shown that every binary cyclic code of length n is completely determined by its generator polynomial, g(x), a divisor of x^n+1. The polynomial x^n+1 with binary coefficients may be factored into n linear factors:

$$(x + \alpha_1)(x + \alpha_2) \ldots (x + \alpha_n)$$

where the roots, α , are elements of some larger field. These n roots can be shown\$to form a cyclic group under the operation multiplication. That is, for some (primitive) root, α , the n roots can be expressed as

$$\alpha^1, \alpha^2, \ldots, \alpha^{n-2}, \alpha^{n-1}, \alpha^n = 1 = \alpha^0$$

The lowest-degree polynomial with binary coefficients which divides x^n+1 and of which α^i is a root is referred to as the minimum polynomial of α^i, and is designated $m_i(x)$. If such a polynomial is considered to have integer coefficients, all coefficients of the polynomial $m_i^2(x) - m_i(x^2)$ are even. In the binary case:

$$m_i^2(x) = m_i(x^2)$$

and if α^i is a root of $m_i(x)$, so are α^{2i}, α^{4i}, α^{8i} . . . The number of roots is the degree of $m_i(x)$. Tables exist for values of n to determine which roots of x^n+1 are roots of a given divisor of x^n+1, or equivalently, which polynomial is the minimum function of a given root of x^n+1.

The BCH bound can now be stated as follows: The minimum distance of the code generated by g(x) must be greater than the largest number of consecutive roots of g(x). (the j roots α^{i+1}, , ..., α^{i+j} are "consecutive" for $0 \leq i \leq n-1$.) Since g(x) has degree n-k, exactly n-k of the roots of x^n+1 are roots of g(x).

Encoding the BCH codes is a straightforward procedure, and obeys the general rules for encoding of cyclic codes. The data polynomial is multiplied by x^{n-k} and divided by the generator polynomial. The residue of this division is added to the data polynomial to form the code word.

5.4.1 Decoding BCH codes

A decoding algorithm for BCH codes that can be implemented with a reasonable amount of equipment or software has been devised by Peterson (43).

The basic decoding problem may be outlined as follows. Consider a code word c(x) transmitted through a noisy channel. Let e(x) denote the error polynomial added to c(x) by the channel. Decoding consists of determining e(x) from s(x), the syndrome calculated from the received polynomial r(x) = c(x) + e(x). Then c(x) is determined by adding e(x) to r(x).

A primitive BCH code has the generator polynomial:

$$g(x) = \text{LCM}\left[m_1(x),\ m_3(x),\ .\ .\ .,\ m_{2t-1}(x)\right]$$

ie., $m_1(x)$, $m_3(x)$, ..., and $m_{2t-1}(x)$ divide $g(x)$ and hence $c(x)$. For the Peterson decoding algorithm it is necessary to compute t partial syndromes by dividing $r(x)$ successively by $m_1(x)$, $m_3(x)$, ..., $m_{2t-1}(x)$. The residue obtained by dividing $r(x)$ by $m_i(x)$ is denoted S_i, and is called the ith partial syndrome. The first partial syndrome can be obtained by dividing $r(x)$ by $m_1(x)$

$$S_1(x) = \text{rem}\ \frac{r(x)}{m_1(x)}$$

But $r(x) = c(x) + e(x)$ and since $m_1(x)$ divides $c(x)$, this gives

$$S_1(x) = \text{rem}\ \frac{e(x)}{m_1(x)}$$

If $e(x)$ has only one nonzero term, that is, $e(x) = x_i$, $0 \leq i \leq n-1$, $S_1(x)$ is the unique residue corresponding to that value of i. If j errors occur, $S_1(x)$ is the sum of the residues corresponding to the various errors. That is, if β_j represents the residue corresponding to the jth error,

$$S_1 = \beta_1 + \beta_2 + \ldots + \beta_j$$

The β are known as the error-locator numbers. If it is possible to determine each of these numbers it is possible to correct the errors since each number is associated with a particular position in the word. The set of residues form a Galois field, and arithmetic operations must be carried out within this field.

There are 2^m residues in the field, each of which is m bits in length; the field consists of 2^m m-tuples obtained by dividing x^i, $0 \leq i \leq n-1$ by $m_1(x)$.

The "elementary symmetric functions", σ , are related to the error locator numbers as follows:

$$\sigma_1 = \beta_1 + \beta_2 + \ldots + \beta_t$$

$$\sigma_2 = \beta_1\beta_2 + \beta_1\beta_3 + \ldots + \beta_{t-1}\beta_t$$

$$\sigma_t = \beta_1\beta_2\beta_3 \ldots \beta_t$$

Having determined the partial syndromes, it is possible to find the elementary symmetric functions. The error locator numbers may then be found by substituting field elements in the equation:

$$\sum(x) = (x+\beta_1)(x+\beta_2) \ldots (x+\beta_t) = 0$$

The binary representations of the exponents of the field elements then point to the bits in error which may subsequently be corrected (44).

5.5 Code Implementation

Several hardware and high-level language implementations of the BCH coding/decoding algorithms have been described in the literature (45-48). This section discusses the microprocessor implementation of these algorithms for real-time operation. It will be shown that the necessary Galois field arithmetic operations may be performed without difficulty using the 6800 microprocessor. Codes of several lengths were investigated; however, special reference is made to the (15,7) dual error correcting code, subsequently employed in the HF modem/codec discussed in chapter 7.

5.5.1 Encoder

It has been shown that a k-bit data block may be encoded into an n-bit cyclic codeword by performing the polynomial operation of equation (5.1) and adding the residue r(x) to the original data block. Encoding therefore simply consists of determining r(x).

The polynomial division was implemented in software (see "BCHCOD", appendix 2), using the "EOR" (exclusive-OR) and the "ASL" (arithmetic shift left) microprocessor instructions to simulate the operation of a shift register implementation.

The (15,7) code has the generator polynomial

$$g(x) = m_1(x)\, m_3(x)$$
$$= (x^4 + x + 1)(x^4 + x^3 + x^2 + x + 1)$$

Encoding a data polynomial d(x) is equivalent to finding

$$c(x) = x^8 d(x) + \text{rem}\ \frac{x^8 d(x)}{g(x)}$$

Multiplication of d(x) by x^8 simply involves shifting the 7-bit data block 8 places to the left; ie. transferring it to the next highest byte.

Modulo-2 division by the generator polynomial was implemented as follows. The binary representation of g(x) is

111010001

This is stored as a left justified double byte which is lined up with the two bytes containing the product $x^8 d(x)$. A bit-by-bit EXclusive OR operation is then performed and the result is shifted left until left justified. The result becomes the new

dividend and the procedure is repeated until a total of eight shifts have been performed. The final result is the residue of the division of $x^8 d(x)$ by $g(x)$, which is added to $x^8 d(x)$ to produce the 15-bit code word.

5.5.2 Decoder implementation

Section 5.4.1 has discussed the operations necessary to implement the Peterson decoding algorithm for BCH codes. Three stages are required:

(1) Computation of the partial syndromes

(2) Calculation of the elementary symmetric functions

(3) Determination of the error locator numbers

Software was written to implement this algorithm for the (15,7) code. (see "BCHDEC", appendix 2) Two partial syndromes are required, S_1, and S_3. These were obtained using polynomial division operations, computed in a similar manner to the encoding procedure. Steps (2) and (3) required operations to be performed in the Galois field. Arithmetic operations in the field may be easily implemented on a microprocessor system as the following section describes.

5.5.3 Galois field operations

Arithmetic operations were to be performed in the Galois field of 2^m (=15) elements. A representation of this field can be formed using the primitive polynomial $m_1(x) = x^4 + x + 1$ (a factor of $g(x)$) and is shown in figure 5.2. The field consists of all polynomials of degree m-1 or less; the field elements can be repesented as 4-bit binary numbers.

α^0	=	1				=	1	0	0	0
α^1	=		α			=	0	1	0	0
α^2	=			α^2		=	0	0	1	0
α^3	=				α^3	=	0	0	0	1
α^4	=	$1 + \alpha$				=	1	1	0	0
α^5	=		$\alpha + \alpha^2$			=	0	1	1	0
α^6	=			$\alpha^2 + \alpha^3$		=	0	0	1	1
α^7	=	$1 + \alpha$			$+ \alpha^3$	=	1	1	0	1
α^8	=	1		$+ \alpha^2$		=	1	0	1	0
α^9	=		α		$+\alpha^3$	=	0	1	0	1
α^{10}	=	$1 + \alpha + \alpha^2$				=	1	1	1	0
α^{11}	=		$\alpha + \alpha^2 + \alpha^3$			=	0	1	1	1
α^{12}	=	$1 + \alpha + \alpha^2 + \alpha^3$				=	1	1	1	1
α^{13}	=	1		$+ \alpha^2 + \alpha^3$		=	1	0	1	1
α^{14}	=	1			$+ \alpha^3$	=	1	0	0	1
α^{15}	=	$1 = \alpha^0$								

FIGURE 5.2 REPRESENTATION OF GALOIS FIELD GF(2^4).

eg. the field element α^8 can be represented as 0101. Methods for implementing operations in the field are described as follows:

(i) Addition. Field elements may be added (modulo 2) term by term in the ordinary way.

(ii) Subtraction. Subtraction of field elements is the same as addition.

(iii) Multiplication. The rule for multiplication is to multiply in the ordinary way, reducing the answer modulo-2 and modulo-$m_1(x)$ to a polynomial of degree m-1 or less. This is done by considering the equation $m_1(x) = 0$ and using the equation to eliminate terms of power greater than m-1. Implementation on a microprocessor can be done using tables of exponents and field elements. Two tables are arranged in memory; the addresses of table 1 correspond to the exponents of α, and the data corresponds to the field elements. The addresses of table 2 correspond to the field elements and the data corresponds to the exponents. To multiply two field elements together, the exponents of the elements are found from table 2, added together modulo-15, and the product field element is determined from table 1. For example, suppose it is required to multiply together the field elements $(1+\alpha)$ and $(1+\alpha^2)$. These elements correspond to the hexadecimal numbers 03 and 05. From table 2, the exponents are found to be 04 and 08. The sum of the exponents is 0C, and the product of the field elements is determined (from table 1) to be OF, which corresponds to the polynomial $1+\alpha+\alpha^2+\alpha^3$.

(iv) division. Division is performed in a similar manner to multiplication, except that the exponents of α are subtracted and not added.

Having computed the partial syndromes S_1 and S_3, it is possible to find the partial syndrome S_2

$$S_2 = S_1^2$$

The error locator numbers are found from:

$$S_1 = \beta_1 + \beta_2$$

$$S_2 = \beta_1^2 + \beta_2^2$$

and the elementary symmetric functions are:

$$\sigma_1 = \beta_1 + \beta_2$$

$$\sigma_2 = \beta_1 \beta_2$$

We find that

$$\sigma_1 = S_1$$

$$\sigma_2 = S_2 + \frac{S_3}{S_1}$$

If field elements are substituted into the equation

$$x^2 + \sigma_1 x + \sigma_2 = 0$$

the errors may be located, and hence corrected. A flowgraph of the complete decoding procedure is shown in figure 5.3. An assembler listing of the implementation of this algorithm is shown in the listing BCHDEC in appendix 2.

5.6 Results

The software implementation of the decoding algorithm was tested by encoding all possible 7-tuples, adding all possible 15-bit error patterns to the resulting codewords, then decoding the corrupted words and comparing with the original codewords to

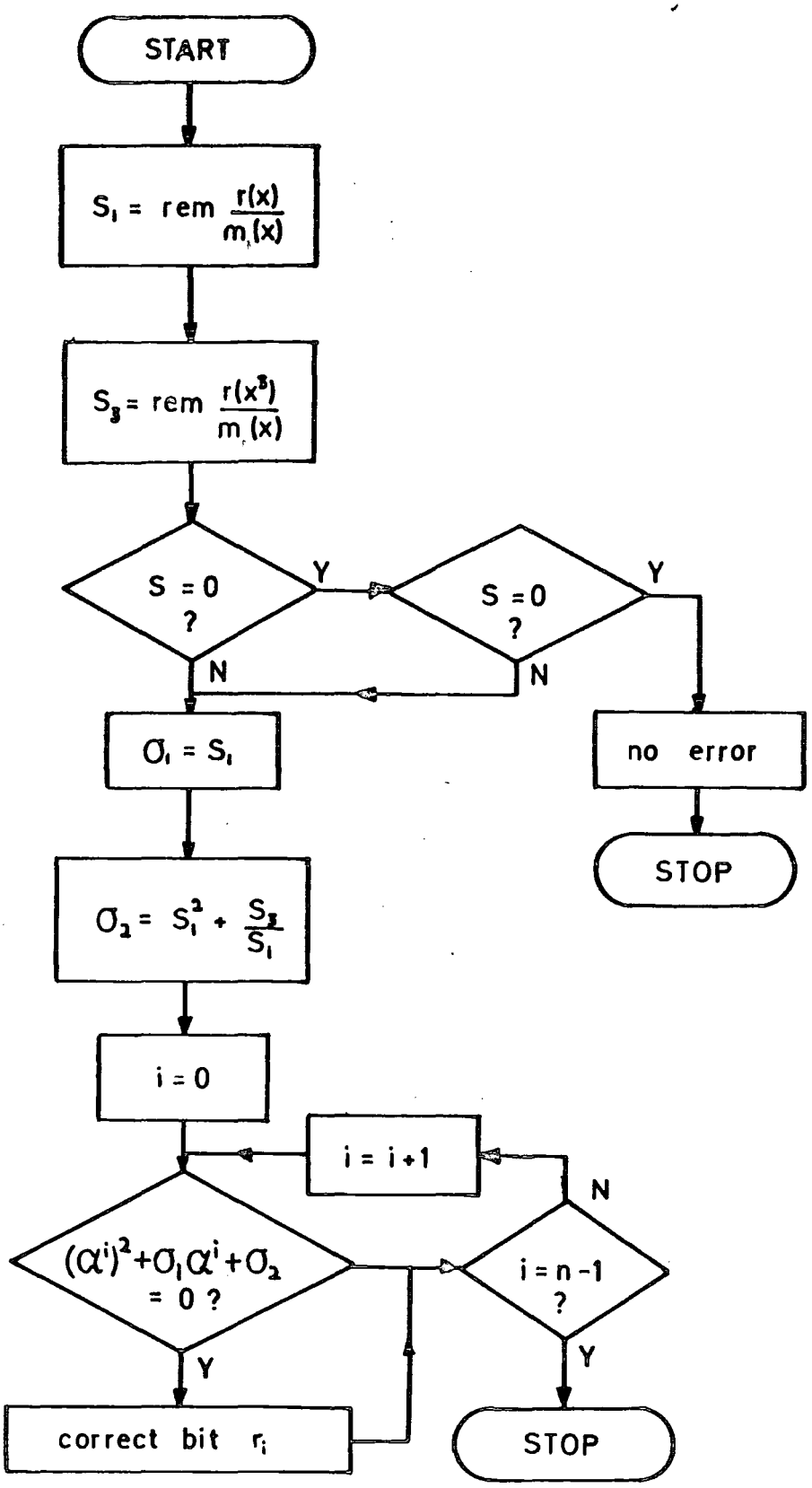


FIGURE 5.3 DECODING PROCEDURE FLOWCHART

determine if the errors had been corrected. For a t-error-correcting code there are:

$$\sum_{i=1}^{t} \frac{n!}{i!(n-i)!}$$

correctable error patterns. For the (15,7) dual-error-correcting code there are 120 correctable patterns of which 15 contain single errors and 105 contain double errors. We may also consider the all-zero 15-tuple to be an allowed error pattern as it results in an uncorrupted word. The test results indicated that all single and double errors were corrected but that correction was not possible for error patterns having a weight exceeding 2.

Measurements were taken of the time required to encode a 7-bit data word and of the time required to decode a received word containing 0, 1, or 2 errors. A slave processor unit was used to time the procedure to the nearest 10 μs. In each case the average time taken for all codewords was measured, together with the best (shortest) and worst times. Best and worst times corresponded to the all-zero and the all-ones codewords respectively. The results are tabulated in table 5.1.

		best	average	worst
	encoding	0.11ms	0.12ms	0.13ms
decoding with:	0 errors	0.31ms	0.59ms	0.85ms
	1 error	1.33ms	1.56ms	1.79ms
	2 errors	1.68ms	1.87ms	2.08ms

Table 5.1. Execution times.

The limitation on the maximum possible data throughput for this coding scheme was determined by the worst case decoding time for 2-error correction. If the maximum decoding time for an n-bit codeword is t_{max} and the code rate is r, the maximum data throughput is:

$$\frac{rn}{t_{max}} = \frac{0.47 \times 15}{2.08 \times 10^{-3}} = 3.36 \text{ kb/s.}$$

Memory utilisation was found to be only 25 bytes for the encoder and 272 bytes for the decoder (including lookup tables).

5.7 Burst error correction

So far, the codes discussed have been cyclic block codes capable of correcting up to t random errors in a block of n bits. Many channels do not exhibit random error characteristics; it is known that HF radio channels are primarily subject to "bursts" of errors. The random error correcting codes described may therefore not always be effective over links using the HF channel. In general, three methods are known for combating the effects of burst errors:

(i) Long, random-error-correcting codes may be designed to have large minimum distance, enabling a large number of errors to be corrected per block.

(ii) Specialised "burst-trapping" codes may be used (49,50).

(iii) Codewords in a random error correcting code may be interleaved to spread the errors over a large number of codewords.

Method (i) necessitates a large amount of hardware or software at the decoder. If the code is to be used purely to

correct burst errors, the random error correcting capability of the code is effectively wasted. This is intuitively obvious if it is considered that there are many more ways of arranging t random errors in a block of n bits than there are ways of arranging t consecutive errors. Codes having large minimum distance have low efficiency; consequently the data rate may be considerably reduced.

Method (ii) requires a good knowledge of the channel statistics. This is not usually possible for HF links (51). If both random and burst errors occur over the channel, such codes may be ineffective.

Method (iii) appears to be the most favourable for data transmission over the HF channel. The method of interleaving can be illustrated as follows. d codewords in a t-error-correcting cyclic (n,k) block code are arranged as rows in a matrix (figure 5.4). The coefficient $c_{i,j}$ represents the jth bit of the ith codeword. The matrix is transmitted as the transpose, ie. column by column, so that a burst of consecutive errors will be spread over more than one codeword. Provided that no more than t errors occur per codeword, all the errors may be corrected at the decoder. The burst correcting ability, b, of the interleaved code is b=dt and d is known as the "interleaving depth". If the length of the error burst is less than dt some random errors may also be corrected. Note that if exactly b errors occur, no more errors may be corrected in that matrix. On average the burst correcting ability of the interleaved code is subject to there being an error-free interval or "guard time" of at least dn bits, thus

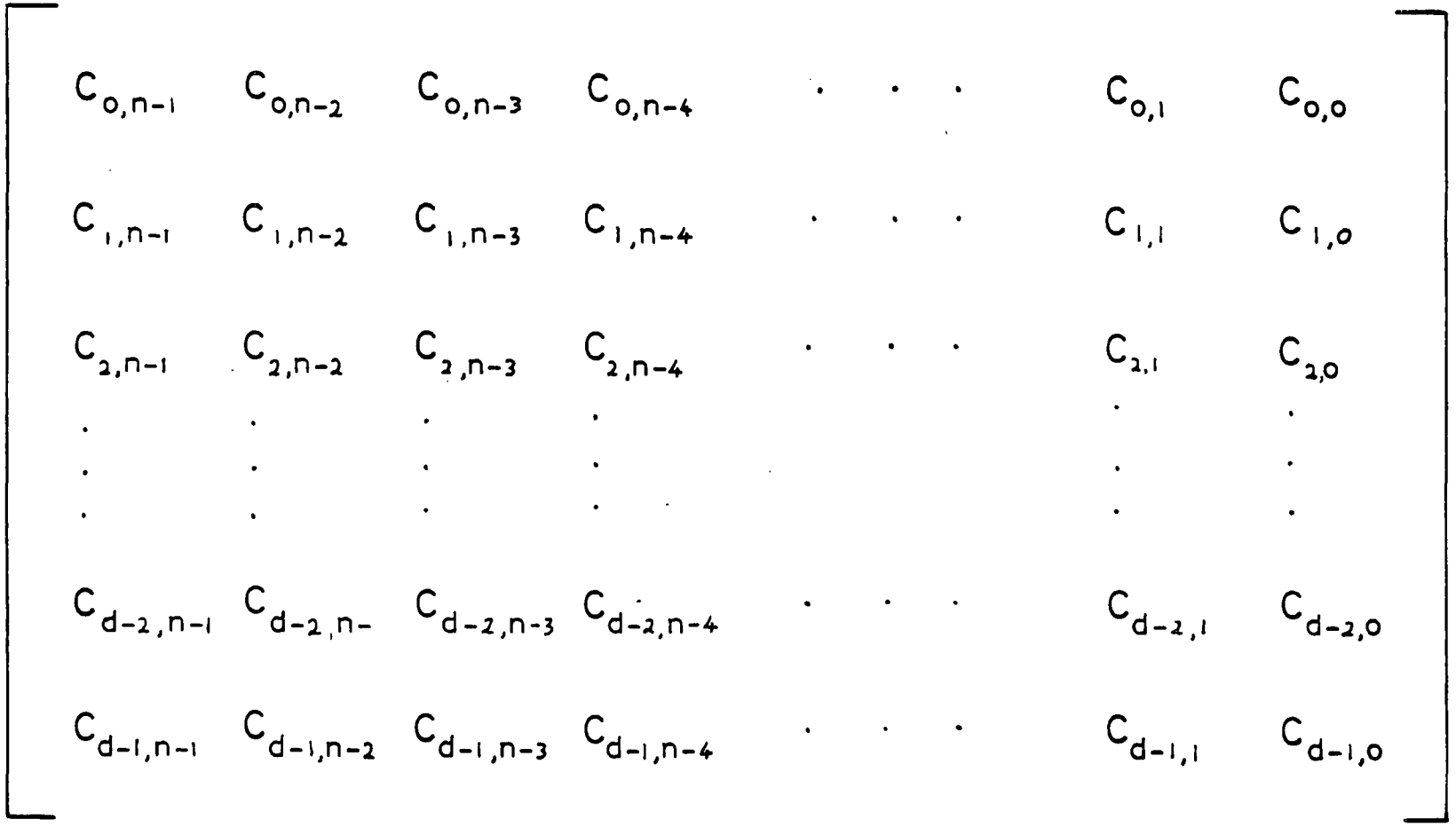


FIGURE 5.4 ILLUSTRATING INTERLEAVING OF CODEWORDS TO COMBAT BURST ERRORS.

ensuring that the number of errors does not exceed the error correcting capablity of the code (52).

Bit interleaving of block codewords may easily be implemented in microprocessor systems using software as will be illustrated in a later chapter.

5.8 Conclusion

This chapter has discussed several properties of cyclic block codes and has shown how these codes may be interleaved to eliminate the burst errors experienced over HF radio links. In particular, a microprocessor implementation of a coding/decoding scheme for the Bose-Chaudhuri-Hocquenghem codes has been discussed in depth. The finite field arithmetic operations required for the decoding algorithm for this code may be easily implemented in real time using conventional microprocessors and allow medium-speed data transmission with a high degree of error protection. Microprocessor implementations of error control are attractive in an adaptive coding environment, as the coding scheme may be changed by simple modifications to the system software.

CHAPTER 6 HF Interference Pattern Measurements

6.1 Introduction

Many of the errors occurring over HF radio data links may be attributed to interference caused by other users of the spectrum (53). A reduction in the error rate is possible if the spectrum of the transmitted signal is arranged to avoid those regions of the channel which contain interference (54). For optimum performance, the spectral distribution of the signal must be adapted to suit the prevailing interference conditions.

An experiment is described in this chapter, in which microprocessor data logging and analysis techniques were used to investigate the fine-grain structure of interference occurring within an HF voice channel. Spectral analysis of the channel was achieved using a charge coupled device to evaluate the chirp-z transform algorithm discussed in section 2.6 of this thesis. The interference measurements were based on an estimation of the power density fluctuations in each of 64 equal-width frequency windows contained within the channel. Results are presented showing the distribution of the occupancy of the windows by interfering signals, and the probability of the occurrence of interference over an interval of time.

6.2 System operation

The system was based around the RC5601 power spectral density board, described in chapter 2. This is an evaluation module based around a CCD quad chirped transversal filter and can be used to calculate power spectral densities from a 512-point

transform by the chirp-z transform algorithm. The module forms a discrete-time spectrum analyser, selecting and outputting the magnitude and frequencies of the spectral components of an analog input waveform. The analysis band extends from zero to the Nyquist frequency (one-half the sample frequency). A mirror image also appears extending from the sample frequency down to the Nyquist frequency. The resolution bandwidth is (1/512) of the sample frequency.

For this application the CCD evaluation module was externally clocked to provide a sampling rate of 6 kHz. The analysis band therefore extended from 0 to 3 kHz, and the resolution bandwidth was 11.7 Hz. The 256 frequency points were reduced to 64 frequency "windows" by combining the energies present in 4 adjacent points. Adjacent windows overlapped because of Hanning windowing of the CCD filters which tended to spread a spectral line over more than 1 point. The overall effect was to produce a series of 64 overlapping bandpass filters. The amplitude-frequency characteristics of the equivalent filters are shown in figure 6.1, and were derived from the following considerations. A computer simulation using the Hanning window equation applied to the CCD transversal filters showed that a sinusoidal input of frequency f_1, coincident with a spectral point, k_1, has its energy distributed as 50% at k_1, 24% at k_1+1, 24% at k_1-1, and 2% outside this region. As shown in the diagram, only 75% of the energy is captured by the filter encompassing k_1, the remainder falling outside the range of the summation. The response at frequency f_1 is therefore $20\log_{10}(0.75) = -2.5$ dB. The overall response of the

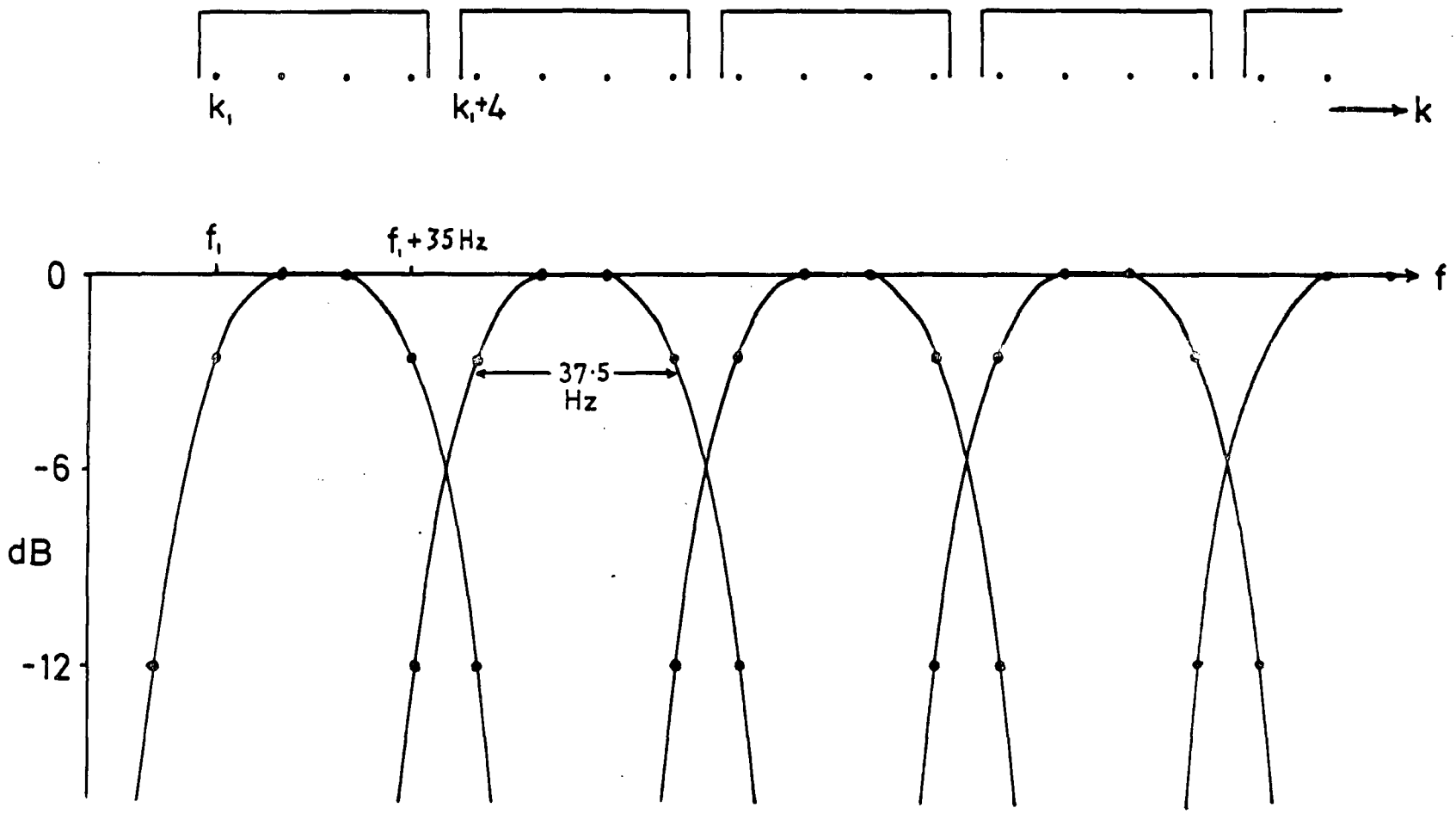


FIGURE 6.1. Equivalent Filter: Amplitude - Frequency Response

equivalent filters was computed from similar considerations.

At a sampling rate of 6 kHz, a complete transform is computed in $(512/6 \times 10^3)$ = 85.3 ms. For this experiment, the power spectrum within the windows was integrated over 2.7s every 16s. This required 32 transforms to be evaluated and averaged at 16s intervals. The output for each spectral point was sampled by an 8-bit analogue-to-digital converter and averaged over the 32 transforms. Each point was then quantised to 16 levels of amplitude, assigned 0-F (hexadecimal). After each 16s interval, the level for each of the 64 points across the frequency range (0-3 kHz) was printed on the terminal, together with the current time of day. Results were also routed to a floppy disc file for subsequent analysis. A 1Hz signal source was used to interrupt the processor at 1s intervals to allow a real-time clock to be maintained within the system.

6.3 Hardware

Figure 6.2 shows a block diagram of the equipment used in the experiment. An inverted-V half-wave dipole antenna was cut to resonate at 4.7925 MHz and was mounted 25m above ground, in a north-south orientation. The antenna was connected to a synthesised communications reciever (the RF-505A), whose audio output was connected to the analogue input of the CCD evaluation module. The CCD module was externally triggered from a signal generator set up to provide pulses of 500 ns duration with a pulse repetition frequency of 6 kHz. The module was synchronised to the microprocessor system by generating an interrupt after each output sample. The output samples were converted to digital

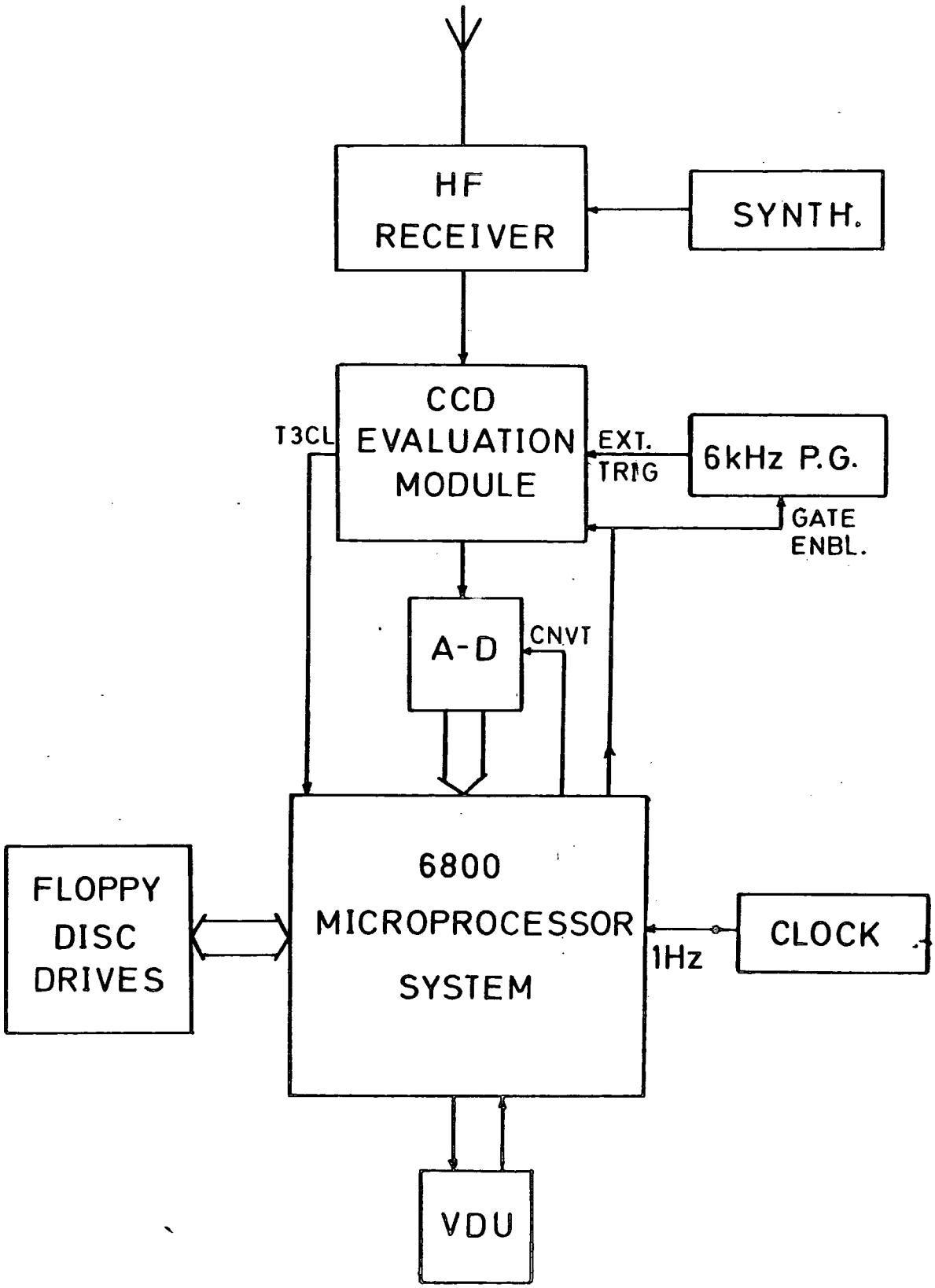


Figure 6.2. Equipment Block Diagram

form using an 8-bit analogue-to-digital converter, and the resulting digital signal was processed as described in the next section. The 1 Hz real-time clock signal was derived from an external digital clock module.

Two 600Ω independent sideband audio outputs are available from the RF-505A receiver. The lower sideband output was selected and was attenuated through a potential divider network to be used as input to the CCD evaluation module. This module accepts an input time signal of 1V peak-peak maximum and produces a spectral output of 4V maximum. The output voltage peak for a constant frequency sinusoidal input will be proportional to the input voltage over this range, as the module computes the rms spectral density of the input signal.

The maximum audio output from the RF receiver was obtained by applying a sine wave from an RF signal generator to the antenna terminal of the receiver and using the BFO to generate a beat note. The attenuator was then adjusted to produce a maximum undistorted output from the evaluation module at the beat note frequency. Following this, the signal generator was removed and the antenna was reconnected to the receiver. The background noise was observed at the output of the CCD evaluation module and was found to produce an average noise output (in the frequency domain) of 0.17V. The dynamic range of the spectral density output was therefore $20\log_{10}(4/0.17) = 27.4$dB. This is somewhat less than the dynamic range of signals received at the front end of the receiver; however, the receiver AGC results in a dynamic range reduction in the audio stages.

The voltage output from the CCD evaluation module was linearly quantised to 16 levels using software (described in the next section). The first quantisation level therefore represented an interference level of $20\log_{10}(0.25/0.17)$ = 3.3dB above the background noise level. This is the interference threshold; ie. the threshold above which a frequency window may be said to contain interference.

6.4 Software

The system software was required to (a) read the sampled analog output data from the CCD evaluation module, (b) find the average magnitude of each group of 4 consecutive frequency bins, (c) average the results over 32 transforms and (d) quantise each averaged magnitude to 16 levels. Routines were also written to store, on flexible disc, the time of day and the number of windows containing any interference.

After system initialisation, the real time clock parameters were set up by entry from the system terminal. At this point, the "in sync" (to the CCD module) and the "gate enable" (to the pulse generator) signals were held at logic '0' by the CB2 output line from the PIA. This caused the 9-bit address counter to be reset. When this line was brought to a '1' state, the sampling clock (6 kHz pulse generator) was enabled, and the address counter was automatically incremented on each clock pulse. The $\overline{\text{T3CL}}$ output from the evaluation module is phase 3 of the 4-phase clock required to transport the charge packets along the CCD. The transition of $\overline{\text{T3CL}}$ coincides with valid output data and was used to cause an IRQ interrupt sequence to be generated by the

PIA.

The first 512 clock periods served merely to load the filter. The Nyquist frequency band was available at the output during the next 256 clock periods, but no output data was sampled until the following 256 clock periods, during which time the signal band was available. It was then necessary to introduce a delay of 256 clock periods, during which time the next Nyquist band appeared at the output. The filter was then operating in a continuous serial mode, outputting one sample in the frequency domain for each sample clocked in the time domain. The process was continued until 32 sets of frequency-domain samples had been collected.

The results from the 32 transforms were averaged and quantised to 16 levels for each frequency window. At each 16s interval, the time of day and the averaged results were printed as a single line of output on the hard-copy terminal. The number of windows in which any interference had been observed was written as a single byte to a flexible disc file, using the disc file management system. The time of day was also stored on the disc, as three binary-coded-decimal numbers. Each disc record therefore contained four bytes of data. The disc file records were subsequently analysed and used to produce the results discussed in the next section.

6.5 Experimental Results

A typical printout of the results obtained during the experiment is shown in figure 6.3. Each printer column to the right of the time corresponds to one of the 64 frequency windows. The magnitude of the response obtained for each window is printed as a single hexadecimal digit in the range 1-F. A magnitude of 0 corresponds to a clear window, for which nothing is printed. Two narrow-band interfering signals can be seen at 1550 Hz and at 2480 Hz. Another (weaker) signal appears at 2300 Hz.

Data for each half-hour interval was analysed and the results were plotted of the number of windows in which **any** interference was detected during the half-hour period. ie. in which the interference level exceeded the previously defined threshold. The results (figure 6.4) are plotted as a percentage of the total number of windows vs. Universal Time for a 24 hour period. For example, between 0330 and 0400, interference was detected in 70% of the total channel bandwidth at some time during the half hour. The data for this experiment was collected during the period 10th-17th March, 1980.

The distribution of interference within the channel was observed to vary considerably over the 24 hours. A high proportion of the channel was occupied during the hours of darkness, when the receiver was subjected to interference from distant sources. A "quiet" interval of about 4 hours was observed around mid-day when no interference was detected.

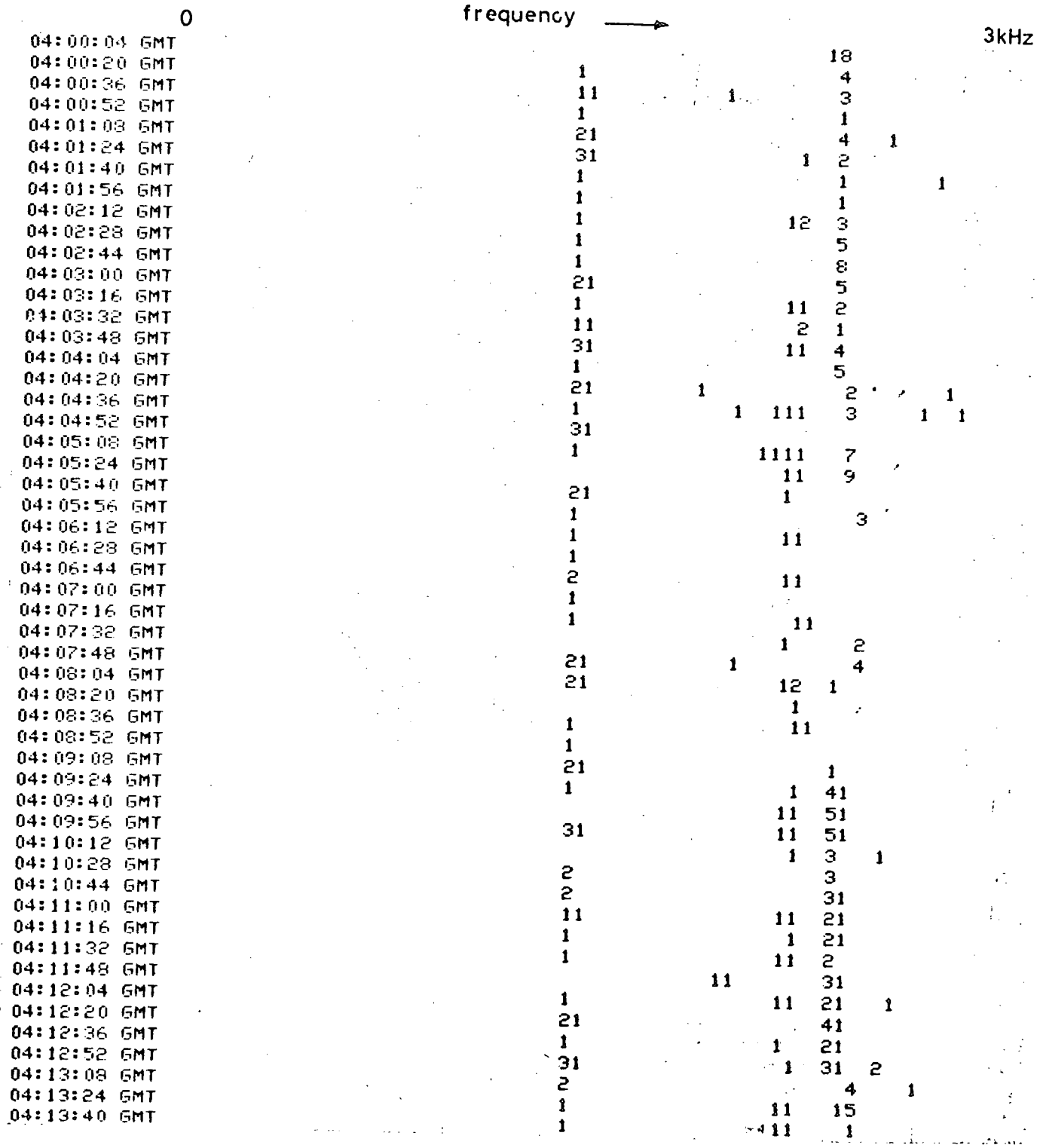


FIGURE 6.3. OUTPUT DATA PRINTOUT.

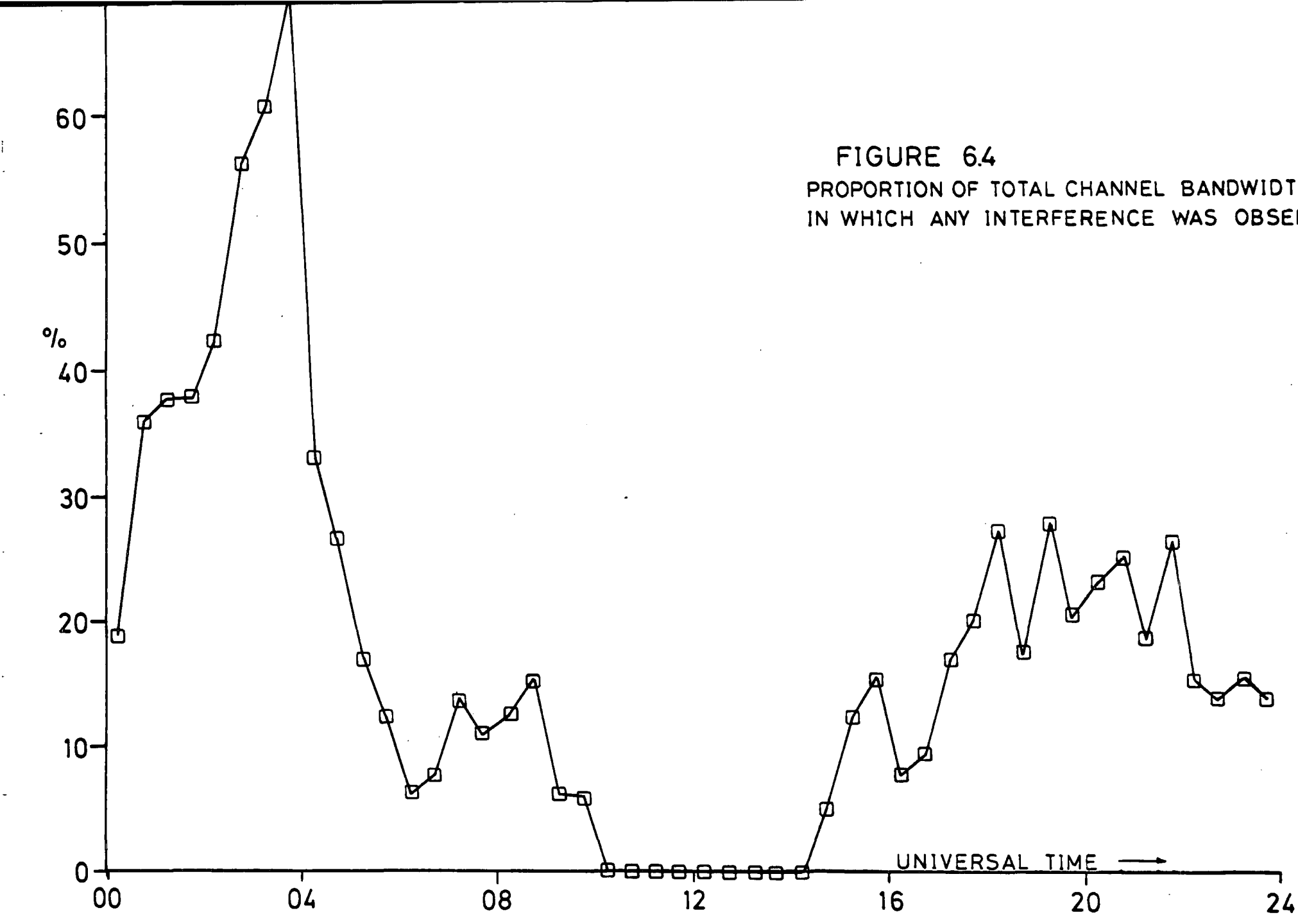


FIGURE 6.4
PROPORTION OF TOTAL CHANNEL BANDWIDTH
IN WHICH ANY INTERFERENCE WAS OBSERVED

The stored data was analysed to determine the proportion of time that the channel was completely free from interference; ie. (subject to propagation being available) the proportion of time that the channel could be used for transmission to its full capacity. The results were plotted for half-hourly intervals and ar shown in figure 6.5. eg. between 0430 and 0500 hrs the channel was found to be completely free from interference for 25% of the time. Interference was heavy during the night amd the channel was occupied to some extent for >95% of the time. There appeared to be a steady improvement in the interference- free time between 0400 and 0630, followed by a slight deterioration and another improvement leading to the 4 hour "quiet" time. This was followed by a gradual deterioration after 1430, probably caused by a lengthening of the skip distance. A noticeable characteristic of the plotted results is that the changes from one half-hour to the next are gradual, with no abrupt transitions.

The stored data was again analysed, to obtain more information on the spectral distribution of the interference within the channel. The proportion of time during which n or more frequency windows were occupied are plotted, for each half-hour interval, in figures 6.6 (n=1,2,3) and 6.7 (n=4,5,6) (both plots are on the same scale). Note that the result for n=1 is simply the inverse of the result of figure 6.5. The percentage falls off rapidly as n increases and becomes very small for $n > 6$. This is indicative of the predominance of narrow-band interference. The results were combined to produce an overall probability of spectral occupancy against bandwidth

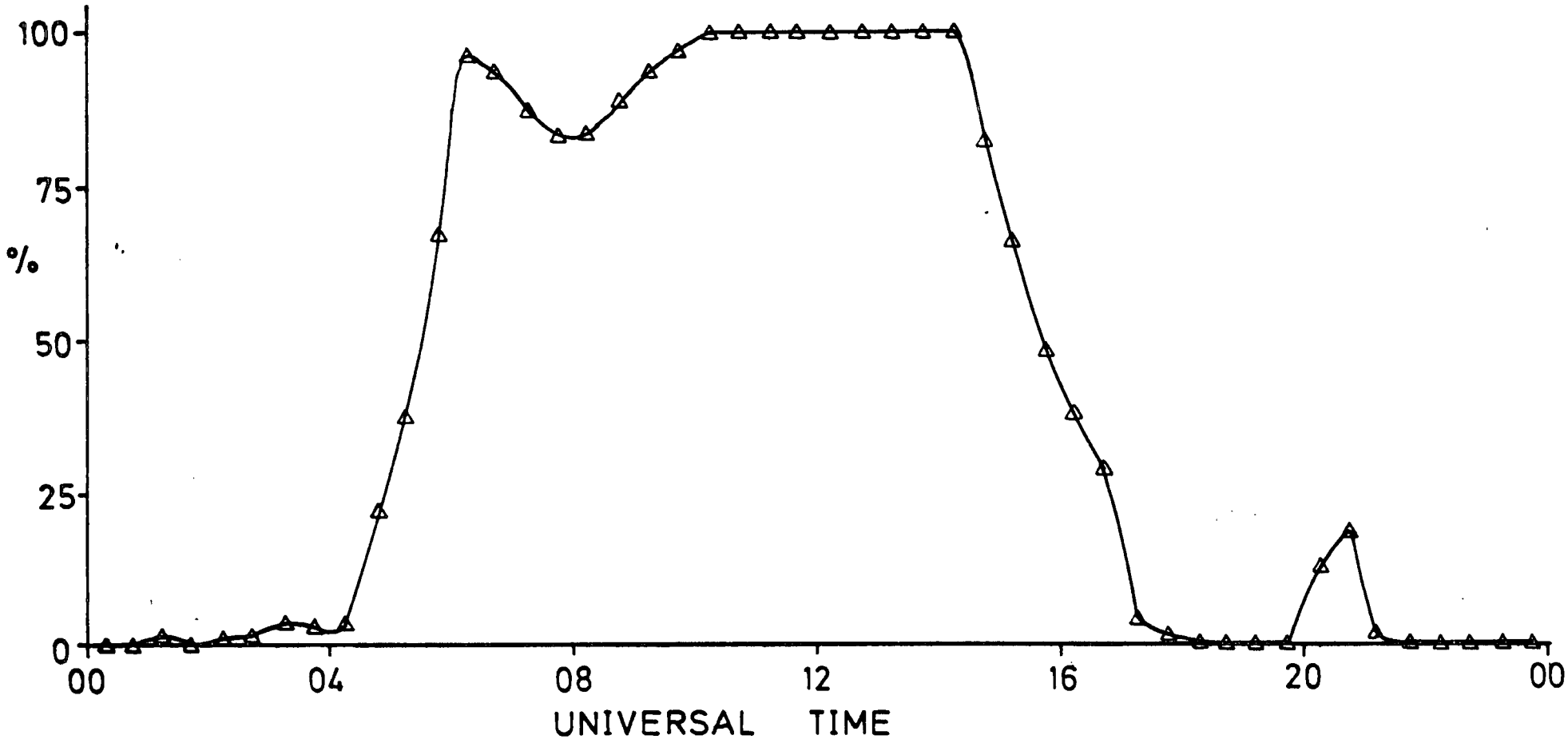


FIGURE 6.5. PROPORTION OF TIME DURING WHICH CHANNEL WAS CLEAR.

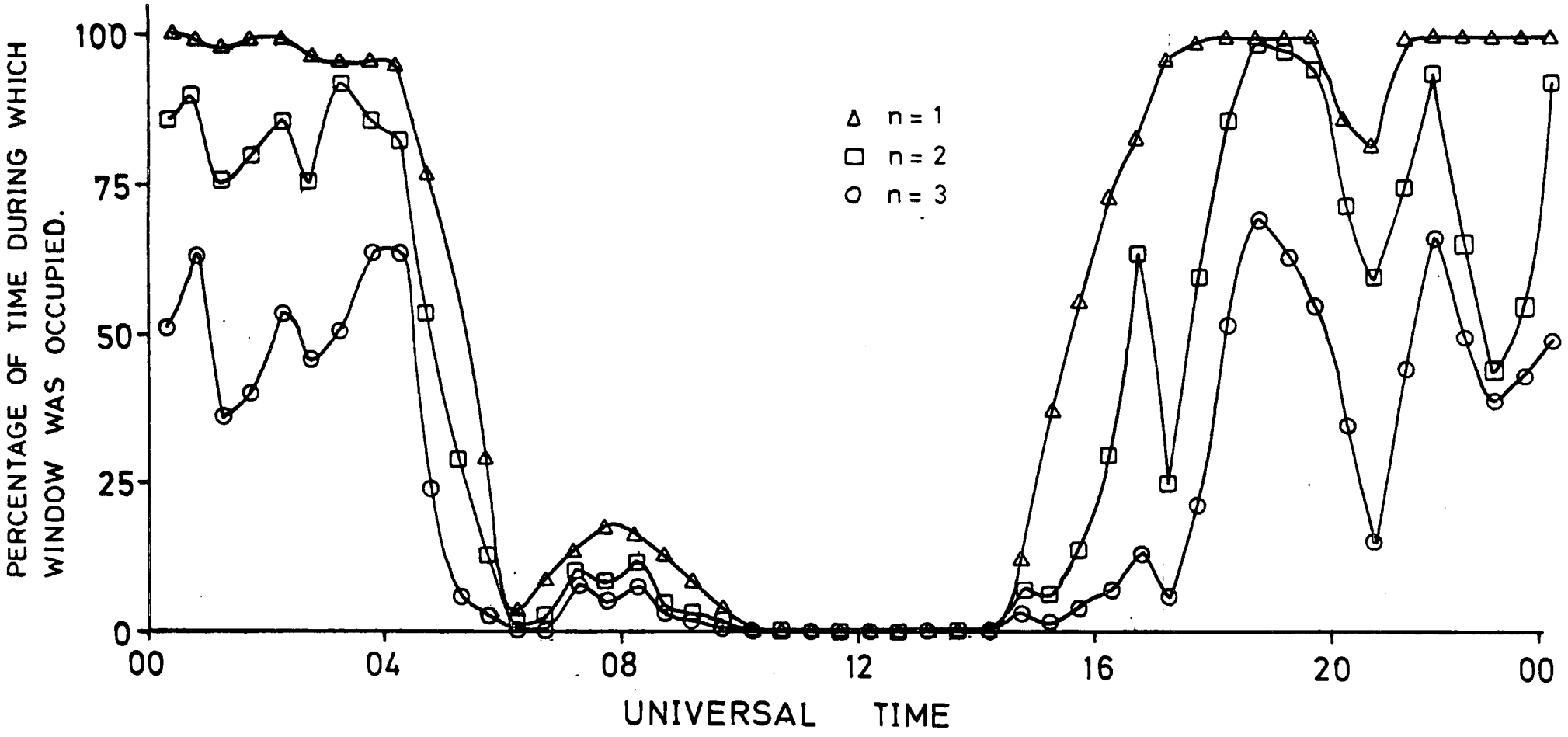


FIGURE 6.6. WINDOW OCCUPANCY

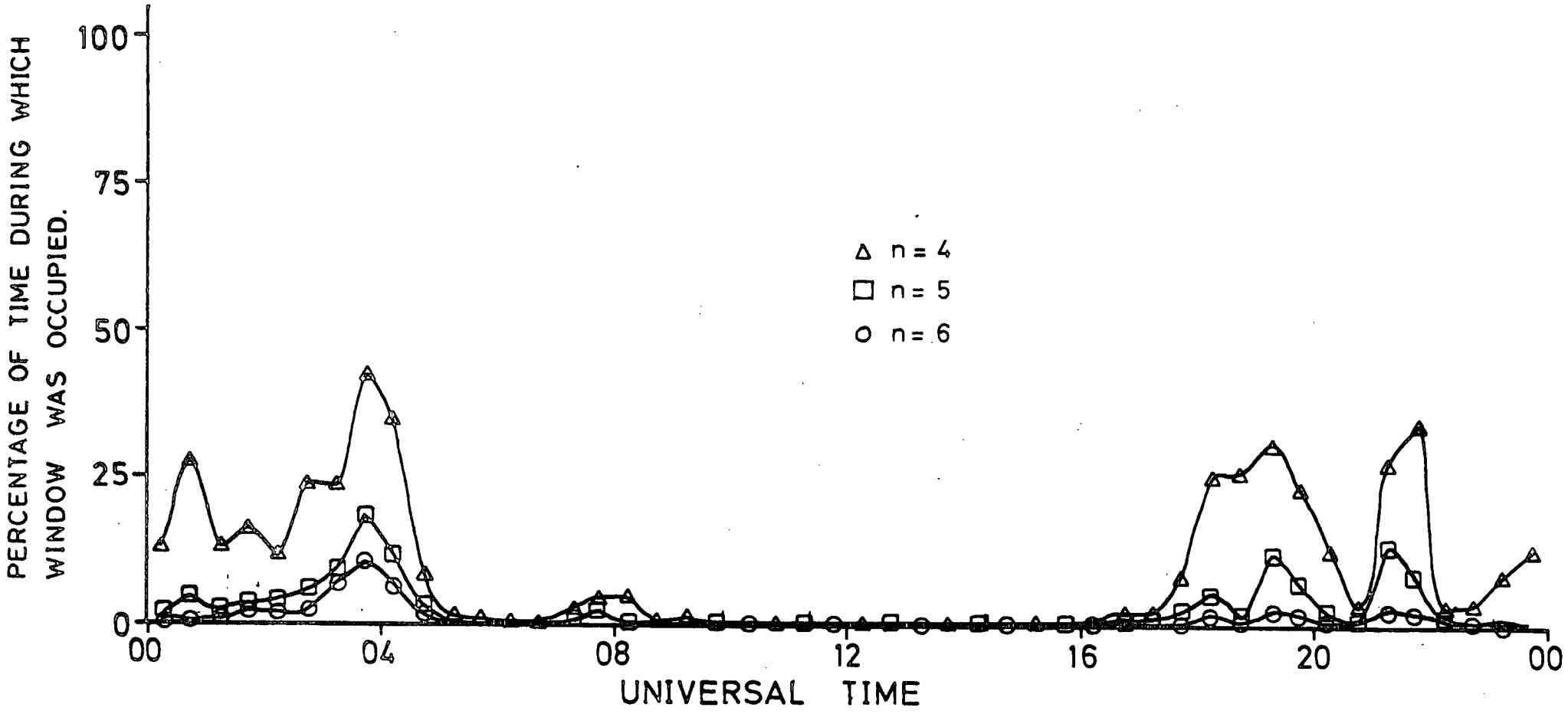


FIGURE 6.7. WINDOW OCCUPANCY

(figure 6.8). Because of the finite width of the frequency windows, the results are plotted as bar graphs, with discrete points at n=0 indicating the probability of a completely free channel. The results appeared to fall into two distinct regions; low interference levels were observed for an eight-hour interval during the daytime; high interference levels were observed during other times, when the skip distance was greater. Results were plotted for these two extremes ie. "day" (0600-1430), and "night" (1430-0600), respectively, and also for a 24 hour average. It can be seen that the probability of a clear channel is much greater during the day, and that at night there is a high probability that narrow-band interference will be present.

6.6 Conclusion

This chapter has described an experiment undertaken, using a microprocessor system and a charge coupled device, to observe the characteristics of interference occurring in an HF radio voice channel. It has been shown that there is a high probability that the channel will contain interference of a narrow-band nature, and that the interference distribution may vary considerably with time, becoming quite severe during the night hours. It is highly probable ($> 97\%$) that the bandwidth of the interference will be narrow (< 200 Hz); however, the distribution of the interference is likely to change rapidly, especially during the night. This confirms the view that benefit may be obtained by using a frequency-agile transmission system to avoid those regions of the transmission channel containing interference.

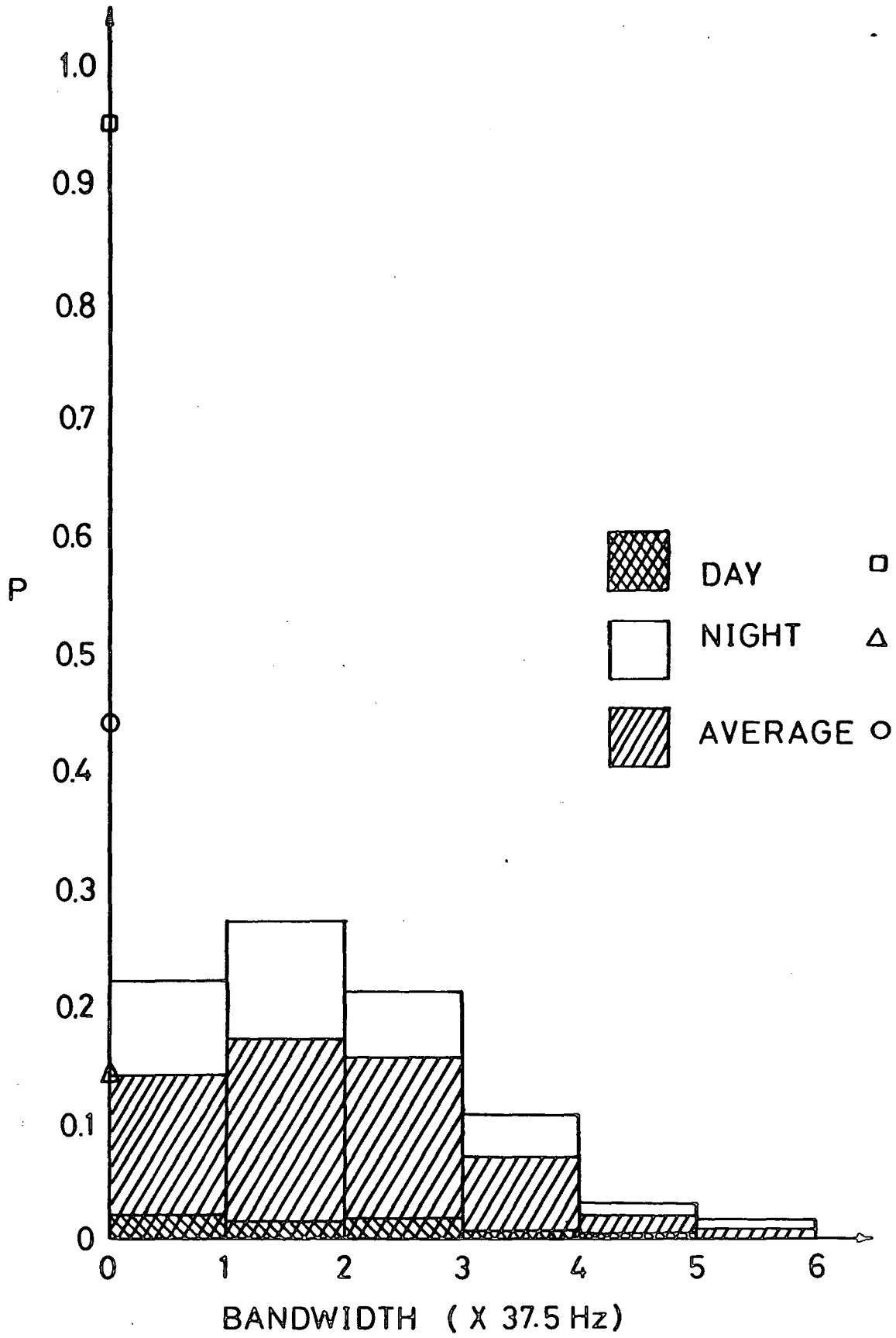


FIGURE 6.8 PROBABILITY OF SPECTRAL OCCUPANCY

CHAPTER 7 The HF Data Modem / Codec

7.1 Introduction

The resurgence of interest in digital communications via HF radio links has led to a renewed search for low-cost modems suitable for data transmission over ionospheric paths. As previously discussed, signals propagated over the HF path are plagued by noise and multipath distortions which may often result in intolerably high error rates (55). The time-varying nature of the disturbances imply that some form of adaptive control is required if the errors are to be eliminated.

The suitability of microprocessor systems for the implementation of signal processing and coding techniques has been demonstrated in chapters 2 and 5. Microprocessors are also extremely attractive for use in an adaptive environment as the course of execution of machine instructions may be made dependent on previous and current events. Non-adaptive modems based on these devices have been demonstrated in the literature (56-59), as have some adaptive schemes which require feedback links from the receiver to the transmitter (60).

This chapter describes the design and development of a medium-speed adaptive HF data modem/codec, based on microprocessor-implemented signal processing and coding techniques, which does not require the use of a feedback link. Results from earlier chapters were considered when formulating the design, which may be implemented at an appreciably lower cost than previous systems.

It has been shown that a primary source of errors over HF radio links is narrow-band interference from other users of the spectrum. The spectral distribution of the modulated output signal from the modem is arranged to occupy the interference-free regions of the radio voice channel. Forward error correction is applied in an attempt to combat transient broad-band disturbances. The modem is described in the published paper of Appendix 3 (reference 61).

7.2 System philosophy

It has been shown (62,63) that the optimum frame (signal element) duration for transmission over a dispersive medium is equal to $\sqrt{L/B}$, where L is the time spread introduced by the medium and B is the frequency spread. For HF channels, both the time spread caused by multipath and the frequency spread caused by Doppler shifting may vary considerably and may depend on the time of day and on the operating frequency. For a medium haul HF link, the time spread may be in the order of several milliseconds, and the Doppler shifting may be of the order of a few Hz. Insertion of typical values in the above formula yields frame rates in the order of tens of Hz. A frame rate of 75 Hz has been found to be a reasonable compromise; the signal element duration is of sufficient length to combat the effects of multipath distortion, yet is short enough to ensure that minimal phase distortion occurs over a single frame. Serial binary-modulated data transmission schemes are therefore limited to a data rate of 75 bps, which does not fully utilise the available bandwidth.

It is possible to increase the transmission rate while preserving the frame rate by time division multiplexing the data for transmission over a number of frequency-parallel subchannels, orthogonally spaced within the voice channel. The data rate is increased by a factor equal to the number of subchannels employed. The spectral distribution of the transmitted signal then depends on the location of the subcarriers. Capacitative coupling and other band-limiting effects inherent in conventional HF radio equipment usually restrict the useable region of an HF voice channel to between 300 Hz and about 2.8 kHz. Sixteen orthogonally spaced slots were allocated within this region for the location of the subcarriers, ie. from 450 Hz to 2700 Hz, with 150 Hz spacings.

It would be possible to utilise the channel to its full capacity by employing subcarriers at all adjacent orthogonal frequency slots. However, results from chapter 6 of this thesis and from work in the literature (53) indicate that there is a high probability that the channel will contain interference, and it is therefore preferable to distribute the signal spectrum in the noise-free regions only. It has been shown that the interference is predominantly narrow-band, but that its distribution may vary rapidly with time. In view of this, a 50% spectral occupancy was adopted, the subcarriers occupying one-half of the available subcarrier slots.

It is also possible to increase the transmission rate while preserving the frame rate by employing a Q-ary modulation scheme (64). In the binary case, it is practical to use only values of

Q which are integer powers of 2, for which the transmission rate is increased by a factor $n=\log_2 Q$. The penalty paid is the increase in vulnerability to noise, which (for PSK) is calculated to be 3dB and 8.5dB for Q=4 and Q=8 respectively (65). 4-phase modulation (QPSK) can therefore be used to to double the effective data rate with only a small reduction in tolerance to additive noise (66) and has been used successfully for data transmission over HF links (67-69). However, because of the phase perturbations observed over such links (70), coherent detection of such a signal becomes virtually impossible and differential phase encoding must be used. This imposes an additional degradation of 2.3 dB for QPSK (71) resulting in a total of 5.3 dB degradation over the optimum coherent biphase detector. The differentially modulated QPSK signal (DQPSK), however, has a spectral occupancy identical to a serially modulated biphase system and gains an advantage in this respect.

The results presented in chapter 6 have indicated that a reduction in the error rate might be obtained if the signal spectrum is adjusted to suit the prevailing channel interference conditions. A system in which the frequencies of the signal carriers are dynamically adjusted is called a "frequency agile" system. A serial in-band frequency-agile FSK modem has been demonstrated by Darnell (54), and has been found to yield improvements over the standard tone allocations. The modem described in this chapter estimates the spectral distribution of the interference present in the voice channel at intervals, and reallocates the distribution of the subcarriers accordingly. An assumption was made that, over medium distances (1000 km. or

less), the interference characteristics at the receiver will be roughly similar to those observed at the transmitter. This assumption was later proved to be correct, as will be demonstrated. The noise estimation may therefore be carried out at the transmitter site, and no feedback link is required. The receiver must be advised, however, as to the new distribution of the transmitted spectrum. This is achieved by transmitting a high-redundancy "advisory sequence" on a set of subcarriers having a fixed frequency allocation.

The receiver demodulator must distinguish the transmitted tones and extract the phase information for each tone. It would be possible to use a bank of conventional narrow-band filters to perform the demodulation process. However a frequency agile system would require a large number of filters, of which only a few would be in use at any one time, making the system uneconomical. It is preferable to use digital processing techniques to implement matched filter detection by means of the Discrete Fourier Transform. Phase information for each subchannel may be obtained by observation of the complex coefficients of the DFT slots corresponding to occupied subchannel frequencies. Comparison detection allows the differential phases (and hence the data) to be extracted without the need for a pilot tone or a locally generated reference phasor.

The design, construction and testing of the transmitter is now described, followed by a discussion of the receiver design.

7.3 Transmitter

A block diagram of the transmitter is shown in figure 7.1, the constituent components of which are discussed in this section. The master processor system includes a 6800 CPU, 4 kbytes of RAM and 2 kbytes of EPROM containing the system software, a listing of which may be found in appendix 2.

The tasks performed by the transmitter fall into four broad categories:

(1) data acquisition

(2) encoding for error control

(3) modulation

(4) channel evaluation & subchannel selection

It was found that a single 6800 microprocessor system was not able to perform all of the required tasks in the available time. One of the main reasons for this was that the modulating waveform must be generated continuously, while simultaneously encoding incoming data. Time-sharing of tasks would inevitably lead to breaks in the analogue signal if this was generated in software. In view of this, a 'master-slave' configuration was used, as described in chapter 4. A 'slave' processor unit was allocated the task of modulation, and the remaining tasks were performed by the 'master'. The advantages of this approach have already been discussed.

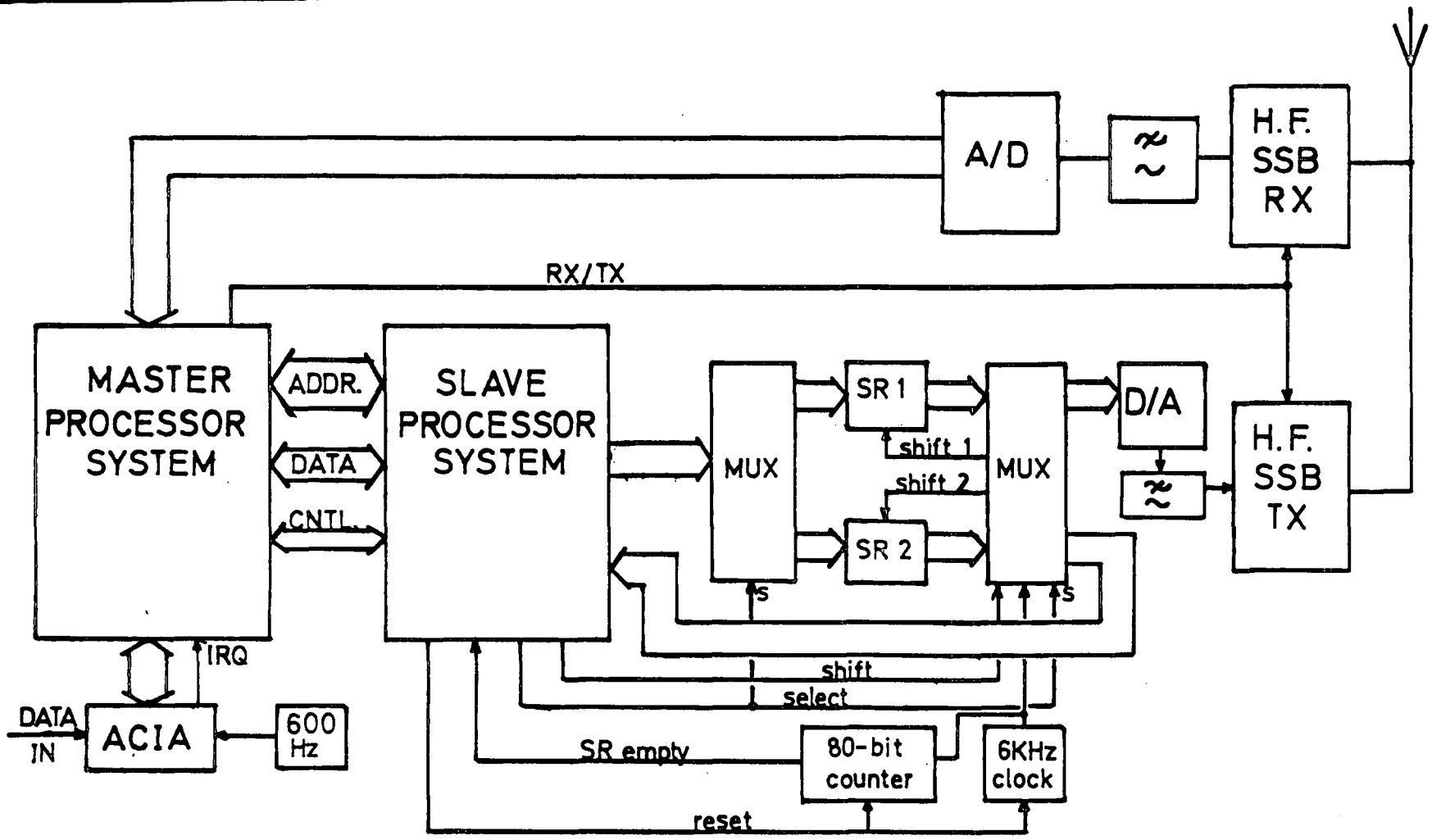


FIGURE 7.1. TRANSMITTER BLOCK DIAGRAM.

7.3.1 Data acquisition

An asynchronous serial interface IC was available which allowed 7-bit characters having start and stop bits to be received and transmitted at a variety of data rates. This IC was used in the system to accept characters at a rate of 600 bps. The interface control register was programmed by the system software such that an interrupt was generated on receipt of a complete character. Each 7-bit character was mapped into a (15,7) BCH codeword (see chapter 5), and was further processed to allow interleaving of 16 codewords along each of the 16 data subchannels. It was necessary to receive 256 characters before transmission could commence (because of the interleaving requirement), after which time characters were accepted continuously via the interrupt driven input.

7.3.2 Encoding for error control

The forward error correction scheme chosen for this system was a (15,7) block code interleaved to depth 16 along each of the 16 data channels. The code has been fully described in chapter 5 of this thesis and the improvement in error rate obtained in practice using this scheme will be demonstrated in the next chapter. Each data character received through the serial interface was coded on receipt and the resulting codeword was loaded into a table of 16 codewords contained in 32 bytes of memory. When the table was full, the bits were interleaved and the whole table was transferred to one of 16 tables containing data ready for transmission via the modem. Each one of these 16 tables corresponded to each of the data channels; a pair of tables therefore corresponded to one subchannel frequency, the

bits in the tables being used to$select the differentiel phase for each signal element. The organisation of the data in time-frequency space is shown in figure 7.2., where $c_{i,j}$ is the jth bit of the ith codeword.

7.3.3 Modulation

Differential four-phase (quatenary) shift keying (DQPSK) was adopted as the modulation scheme for the system. The z-plane representation of a suitable phase encoding scheme is shown in figure 7.3 where the axes form the decision thresholds and the bit-pairs corresponding to the phases are arranged in a Gray code around the unit circle to minimise the bit error probability.

Each subcarrier conveys two bits of information per signal element and there is a total of eight subcarriers to be located in a possible 16 orthogonally spaced frequency slots. 16 bits are therefore transmitted per signal element, which are divided into 8 pairs; a pair of bits determines the phase on the corresponding sub-channel. The signal over one element may be described as:

$$f(t) = \sum \cos (\omega k_c t + \phi_c) , \quad 0 \leq t < 13.33 \text{ ms.}$$

where $\omega = 2\pi$ 150 Hz, k_c is an integer$between 3 and 18 and ϕ_c is the phase corresponding to subchannel c (c is an integer). ϕ_c may have values $\pi/4$, $3\pi/4$, $5\pi/4$, $7\pi/4$.

The composite modulating signal was to be generated digitally, then low-pass filtered to confine the spectrum to the

SUBCHANNEL NUMBER	DATA BITS
7	$C_{255,14}, C_{254,14}, \ldots C_{240,14}, C_{255,13}, C_{254,13}, \ldots C_{240,0}$
	$C_{239,14}, C_{238,14}, \ldots C_{224,14}, C_{239,13}, C_{238,13}, \ldots C_{224,0}$
6	$C_{223,14}, C_{222,14}, \ldots C_{208,14}, C_{223,13}, C_{222,13}, \ldots C_{208,0}$
	$C_{207,14}, C_{206,14}, \ldots C_{192,14}, C_{207,13}, C_{206,13}, \ldots C_{192,0}$
5	$C_{191,14}, C_{190,14}, \ldots C_{176,14}, C_{191,13}, C_{190,13}, \ldots C_{176,0}$
	$C_{175,14}, C_{174,14}, \ldots C_{160,1}, C_{175,13}, C_{174,13}, \ldots C_{160,0}$
4	$C_{159,14}, C_{158,14}, \ldots C_{144,14}, C_{159,13}, C_{158,13}, \ldots C_{144,0}$
	$C_{143,14}, C_{142,14}, \ldots C_{128,14}, C_{143,13}, C_{142,13}, \ldots C_{128,0}$
3	$C_{127,14}, C_{126,14}, \ldots C_{112,14}, C_{127,13}, C_{126,13}, \ldots C_{112,0}$
	$C_{111,14}, C_{110,14}, \ldots C_{96,14}, C_{111,13}, C_{110,13}, \ldots C_{96,0}$
2	$C_{95,14}, C_{94,14}, \ldots C_{80,14}, C_{95,13}, C_{94,13}, \ldots C_{80,0}$
	$C_{79,14}, C_{78,14}, \ldots C_{64,14}, C_{79,13}, C_{78,13}, \ldots C_{64,0}$
1	$C_{63,14}, C_{62,14}, \ldots C_{48,14}, C_{63,13}, C_{62,13}, \ldots C_{48,0}$
	$C_{47,14}, C_{46,14}, \ldots C_{32,14}, C_{47,13}, C_{46,13}, \ldots C_{32,0}$
0	$C_{31,14}, C_{30,14}, \ldots C_{16,14}, C_{31,13}, C_{30,13}, \ldots C_{16,0}$
	$C_{15,14}, C_{14,14}, \ldots C_{0,14}, C_{15,13}, C_{14,13}, \ldots C_{0,0}$

t ⟶

FIGURE 7.2. DATA ORGANISATION IN TIME-FREQUENCY SPACE.

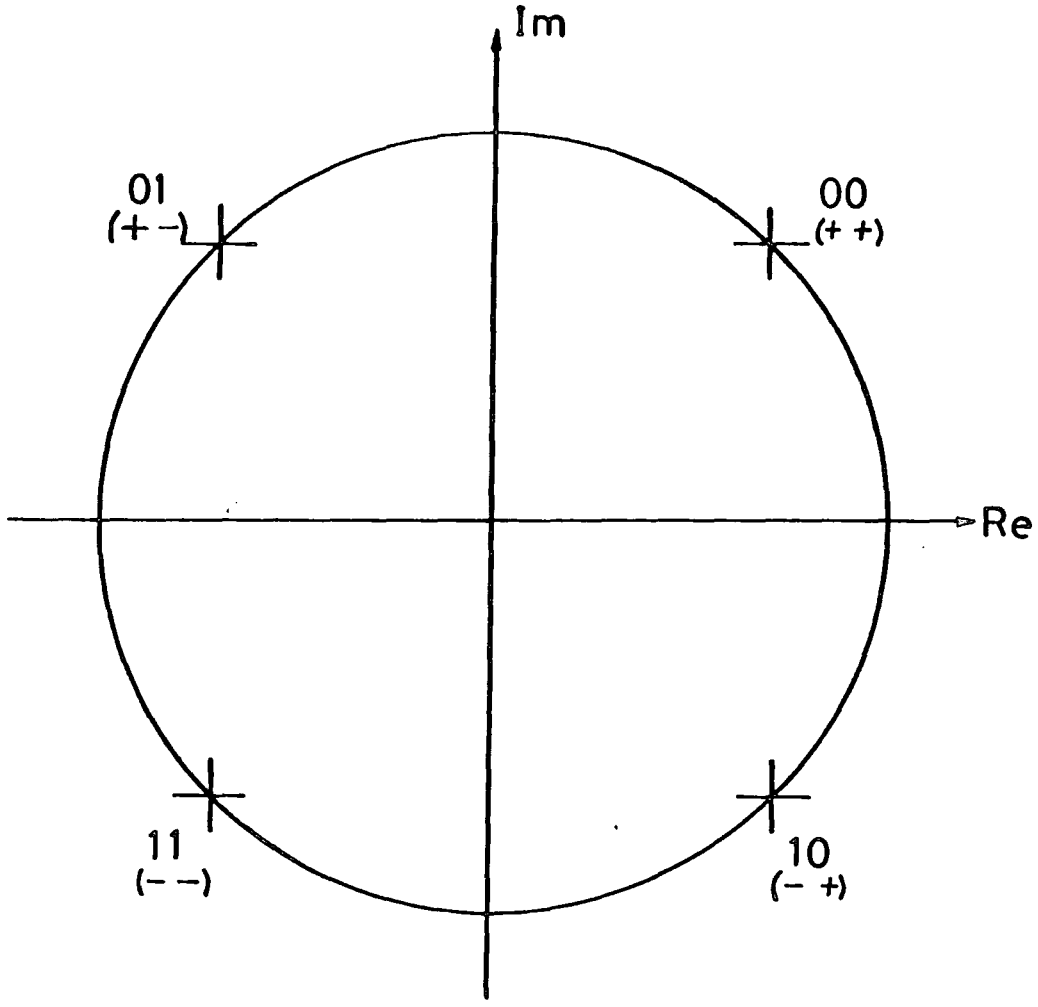


FIGURE 7.3. Z-PLANE REPRESENTATION OF QPSK.

required signal band. A lookup table stored in the slave memory was used to generate samples of this signal. The table consists of N samples of a cosine waveform, equally spaced in time, each of which is stored as an 8-bit 2's complement number. If consecutive samples from the table are output at a sampling rate of f_s, the fundamental frequency of the resulting waveform is f_s/N. Multiples of this frequency may be generated by stepping through the table at different rates, so that if the step length is l, the frequency generated is lf_s/N. The table may be regarded as circular; the index of the current sample is always calculated modulo-N. Nyquist's sampling theorem requires that at least 2 samples per waveform cycle must be available to define a frequency, implying that the sampling rate must be at least twice the frequency of the highest component. For this application a sampling rate of 6kHz is sufficient to guarantee this condition, and if frequencies are to be generated in multiples of f_c = 150 Hz (to satisfy the orthogonality constraint), the length of the lookup table must be $N = f_s/f_c = 40$ samples.

The frame rate of the transmitted signal is one-half of the frequency spacing so that a complete frame is generated using 80 samples at the 6kHz sampling rate. The phase of a particular subchannel is determined by the starting point in the lookup table. Phases of $n\pi/4$ (n = 1,3,5,7) are required, corresponding to the dibits 00,01,11,10 respectively. The required phase is generated by starting at sample number $nN/8 = 5n$.

As an example of the preceding discussion, suppose it is required to generate a subchannel on 900 Hz having a phase of

$5.\pi/4$ radians. The step length is $900/f_c = 6$ and the starting point in the table is 25. The first 10 indices of the 80 samples required to generate the subchannel are then:

25,31,37,3,9,15,21,27,33,39

Modulator circuit

Consideration was given to the possibility of generating the composite multitone waveform using a purely software approach. One method would be to use an interrupt processing routine to generate a new sample of this waveform at each sampling period. However, the overhead required in adopting this method represented a considerable proportion of the overall processing time, and it was not possible to complete the required processing in the available baud time of 13.33 ms. Additional hardware was therefore designed and built to reduce the computational requirements of the modulation process. Operation of the modulator hardware is described in this section. A circuit diagram is shown in figure 7.4.

The modulator circuit is interfaced to the slave processor via two PIAs (Peripheral Interface Adapters). That to the left of the circuit diagram is referred to as the 'input PIA' and that to the right is the 'output PIA'. Twelve of the 16 I/O port lines from the input PIA are connected to the inputs of four hex tri-state buffers, arranged as two pairs. The buffer outputs are connected to the inputs of six quad 80-bit static shift registers, arranged as two sets of three, subsequently referred to as SR1 and SR2. The shift register outputs are connected to six 8-to-4 line multiplexers, also arranged as two sets of three. The outputs

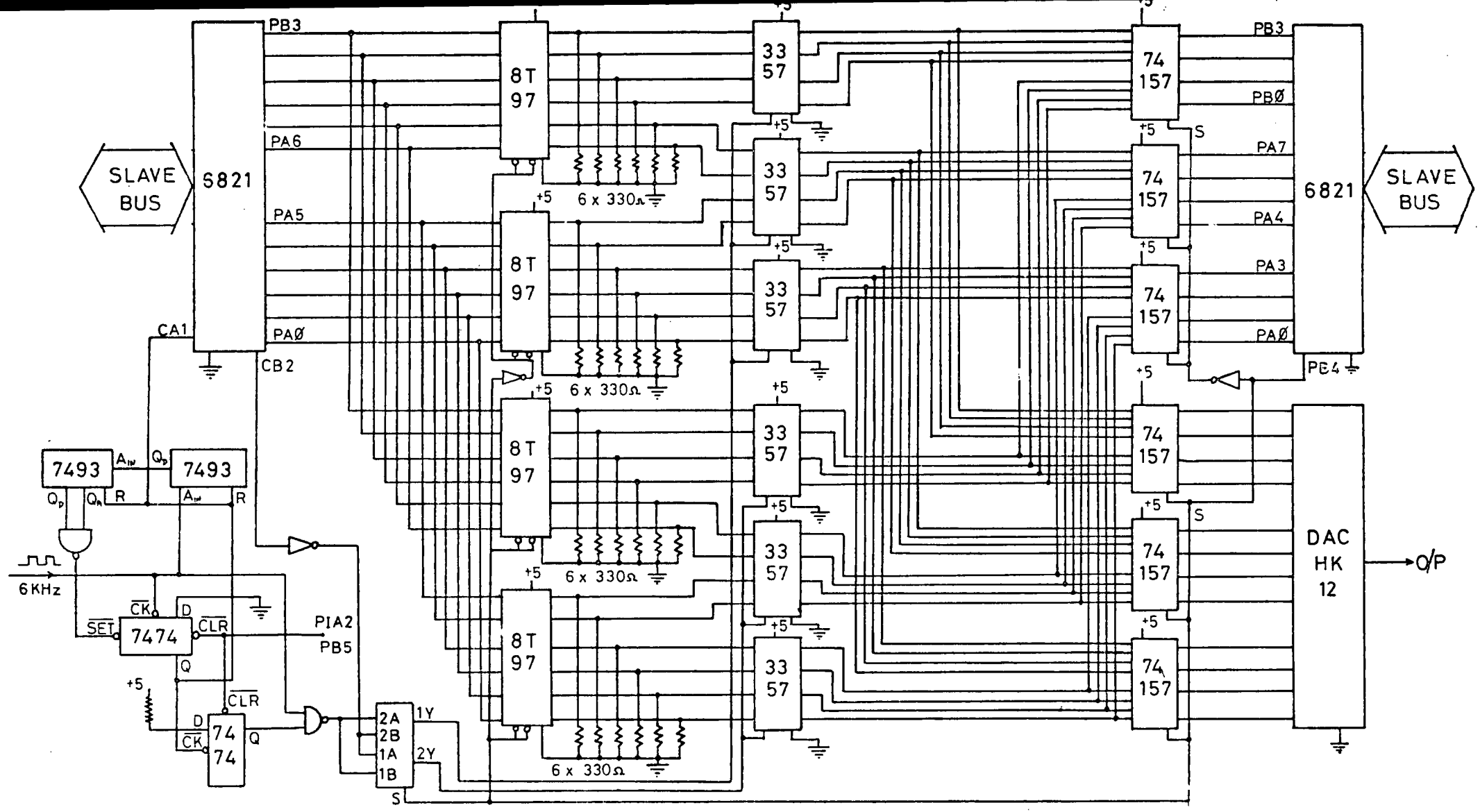


FIGURE 4.4. MODULATOR CIRCUIT DIAGRAM

from the first set of multiplexers are connected to twelve I/O lines of the output PIA, and the outputs from the second set are connected to the inputs of a 12-bit digital to analogue converter. The output from a 6kHz square wave generator (implemented using a crystal oscillator and a divider circuit) is connected to the input of two cascaded 4-bit binary counters. The Q_A and Q_D outputs from the most significant counter are NANDed together and are used to 'set' a D-type bistable on the count of 80. The output from this bistable then resets the counters and is used to interrupt the slave processor unit via the CA1 line on the input PIA. The remaining bistable in the 7474 IC is used to enable the 6kHz sampling clock to the input of a multiplexer. The inverted signal from the CB2 line of the input PIA is also connected to the input side of this multiplexer. The multiplexer is configured so that the 1Y output is derived from the CB2 line while the 2Y output is derived from the sampling clock, and vice versa.

The enable lines to the data buffers are inverted with respect to each other so that while one pair is enabled, the other is not. The outputs of the pair that is not enabled are pulled to logic '0' by the resistors. The select lines on the multiplexers are also inverted with respect to each other so that the outputs from the shift registers whose inputs are derived from the enabled data buffers are routed to the output PIA, while the outputs from the other shift registers are routed to the D/A converter. The 'shift' signal for the former is obtained from the CB2 output line of the input PIA, via the remaining multiplexer, and the shift signal for the latter is derived from the sampling clock.

After loading, the slave processor memory contains the program for control of the modulation, a 40-point cosine lookup table, and two 8-byte tables containing the step lengths and starting points for each subchannel. Operation of the circuit is as follows:

The multiplexers are switched so that the PIA output lines are routed into the input of register SR1 and the outputs of SR1 are routed into the input PIA. Samples for the first subchannel are selected from the lookup table using the corresponding step length and starting point. Each sample is written into the eight least significant bits of SR1; the register is full after 80 "write" instructions. The first sample entered is then read from the shift register output, added to the first sample of the next subchannel and the result (now 9 bits) is written back into the shift register. After 80 shifts the register contains the sum of the samples for the first two subchannels. The procedure is continued for all eight subchannels. The multiplexers are then switched so that the "shift" line to SR1 is derived from the 6kHz clock and the shift line for SR2 is derived from CB2. The output from SR1 is now routed to the D to A converter and the register contents are shifted out at the sampling rate of 6 kHz. During this time the register SR2 is loaded with samples for the next frame in a similar manner. When SR2 is full, the multiplexers are again switched and the contents are routed to the D to A converter. During one frame, therefore, the contents of one register are shifted out to the converter at the sampling rate while the slave processor is constructing samples for the next frame using the other shift register. When all the samples in a

register have been shifted out, the counter is used to interrupt the processor which then switches the multiplexer select line and begins to construct the next frame.

The frequencies and phases of the subchannels for each frame are determined by the contents of two 8-byte tables which contain the step lengths and starting points respectively. There are actually two pairs of tables; the contents of one pair are used by the slave to compute samples for the frame currently under construction, while parameters are loaded into the other pair from the master for use by the slave during the following element. During one frame, therefore, the master has control of one pair of tables while the slave has control of the other. Upon receipt of an interrupt from the slave (indicating completion of transmission of a frame), control of the tables is reversed.

To summarise the preceding discussion, three processes occur simultaneously during one signal frame. Samples for the current frame are shifted into the D-A converter from one set of shift registers by the 6kHz sampling clock; samples for the next frame are computed by the slave in the second set of shift registers with phase and frequency information obtained from one pair of tables. Frequency and phase information for the third frame is passed from the master into the second pair of tables in the slave.

7.3.4 Channel evaluation & subchannel selection

Following each message transmission, the HF transceiver is switched to the 'receive' mode (by means of a Tx/Rx reed relay), and the audio output of the receiver is sampled by the A-D converter. 64 samples are acquired at a sampling rate of 9.6 KHz and an in-place FFT is computed (see chapter 2). The resulting DFT frequency slots are therefore in multiples of 150 Hz, and the power spectrum of the slots from 450 Hz to 2700 Hz inclusive (the 16 available subchannel slots) is estimated from the phasor magnitudes. The sampling and transformation processes are repeated 8 times and the resulting power spectra are averaged to provide a reasonable estimation of the noise present in each subchannel slot over an observation interval of approximately 3.5s. This represents only 2% of the overall transmission time and therefore does not significantly affect the data rate. A number sorting routine (CHSORT) is then used to select the eight "quietest" slots for subsequent transmission. The quietest slots are taken to be those those eight which exhibit the least average interference from the total of sixteen available slots. A typical distribution of the signal spectrum is illustrated in figure 7.5.

In order that the receiver may be advised as to the subchannel frequencies to be used for subsequent data transmission, a coded sequence containing this information is transmitted immediately before the frequency change is effected. The sequence comprises eight (15,7) BCH codewords; the information section of each is formatted as follows:

$$n_2\ n_1\ n_0\ s_3\ s_2\ s_1\ s_0$$

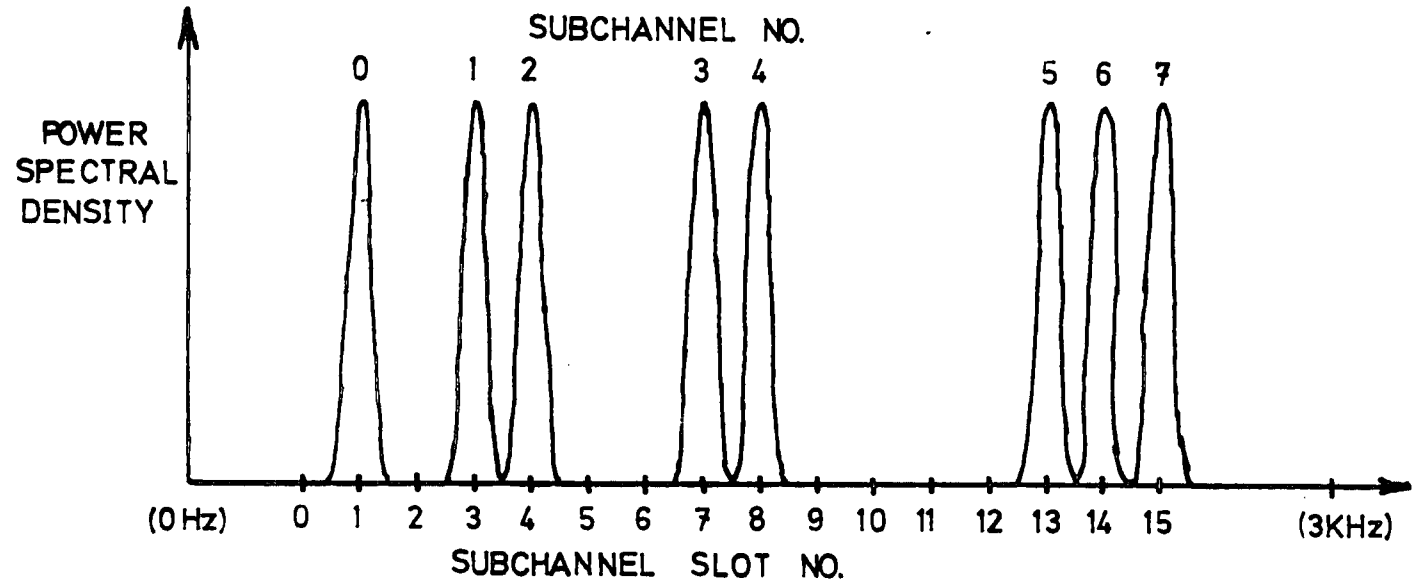


FIGURE 7.5. TYPICAL SPECTRAL DISTRIBUTION OF SUBCHANNELS.

The vector $\mathbf{n} = (n_2 n_1 n_0)$ represents a subchannel number in the range 0-7 and the vector $\mathbf{s} = (s_3 s_2 s_1 s_0)$ represents a subchannel of frequency (450 + (s x 150)) Hz. A code vector **c** is found from the matrix operation:

$$\mathbf{c} = [\mathbf{ns}]\mathbf{G}$$

where G is the code generator matrix.

The code is fully described in chapter 5. The set of 8 code vectors are arranged in time-frequency space as illustrated in figure 7.6, where $c_{i,j}$ is the jth bit of the ith codeword. It can be seen that pairs of codewords are interleaved to depth 2 along each data channel allowing correction of 4 errors per data channel. Because of the 4-phase modulation scheme there are two data channels per subchannel frequency, resulting in a total of 16 data subchannels. For each signal element, the differential phase transmitted on a subchannel is determined by the corresponding dibit. There is a four-fold spectral redundancy in the transmitted data which compensates for any narrow-band interference which may be present. The advisory sequence is always transmitted on a set of subchannels having a fixed frequency allocation. Transfer to the new subchannels occurs immediately after the advisory sequence transmission. If a fixed frequency allocation was not used, a situation might arise where an error occurs in the decoding of the advisory sequence, resulting in the loss of all subsequent messages. The fixed allocation allows the receiver to recover after a single message block, since the location of the subchannels for the advisory sequence is always known.

SUBCHANNEL FREQUENCY

2550	$C_{7,14}$	$C_{6,14}$	$C_{7,13}$	$C_{6,13}$	...	$C_{7,0}$, $C_{6,0}$
	$C_{5,14}$	$C_{4,14}$	$C_{5,13}$	$C_{4,13}$	...	$C_{5,0}$, $C_{4,0}$
2250	$C_{3,14}$	$C_{2,14}$	$C_{3,13}$	$C_{2,13}$	...	$C_{3,0}$, $C_{2,0}$
	$C_{1,14}$	$C_{0,14}$	$C_{1,13}$	$C_{0,13}$	...	$C_{1,0}$, $C_{0,0}$
1950	$C_{7,14}$	$C_{6,14}$	$C_{7,13}$	$C_{6,13}$	...	$C_{7,0}$, $C_{6,0}$
	$C_{5,14}$	$C_{4,14}$	$C_{5,13}$	$C_{4,13}$	...	$C_{5,0}$, $C_{4,0}$
1650	$C_{3,14}$	$C_{2,14}$	$C_{3,13}$	$C_{2,13}$	...	$C_{3,0}$, $C_{2,0}$
	$C_{1,14}$	$C_{0,14}$	$C_{1,13}$	$C_{0,13}$	...	$C_{1,0}$, $C_{0,0}$
1350	$C_{7,14}$	$C_{6,14}$	$C_{7,13}$	$C_{6,13}$	...	$C_{7,0}$, $C_{6,0}$
	$C_{5,14}$	$C_{4,14}$	$C_{5,13}$	$C_{4,13}$	...	$C_{5,0}$, $C_{4,0}$
1050	$C_{3,14}$	$C_{2,14}$	$C_{3,13}$	$C_{2,13}$	...	$C_{3,0}$, $C_{2,0}$
	$C_{1,14}$	$C_{0,14}$	$C_{1,13}$	$C_{0,13}$	...	$C_{1,0}$, $C_{0,0}$
750	$C_{7,14}$	$C_{6,14}$	$C_{7,13}$	$C_{6,13}$	...	$C_{7,0}$, $C_{6,0}$
	$C_{5,14}$	$C_{4,14}$	$C_{5,13}$	$C_{4,13}$	...	$C_{5,0}$, $C_{4,0}$
450	$C_{3,14}$	$C_{2,14}$	$C_{3,13}$	$C_{2,13}$	...	$C_{3,0}$, $C_{2,0}$
	$C_{1,14}$	$C_{0,14}$	$C_{1,13}$	$C_{0,13}$	...	$C_{1,0}$, $C_{0,0}$

t ⟶

FIGURE 7.6. ADVISORY SEQUENCE FORMAT IN TIME-FREQUENCY SPACE.

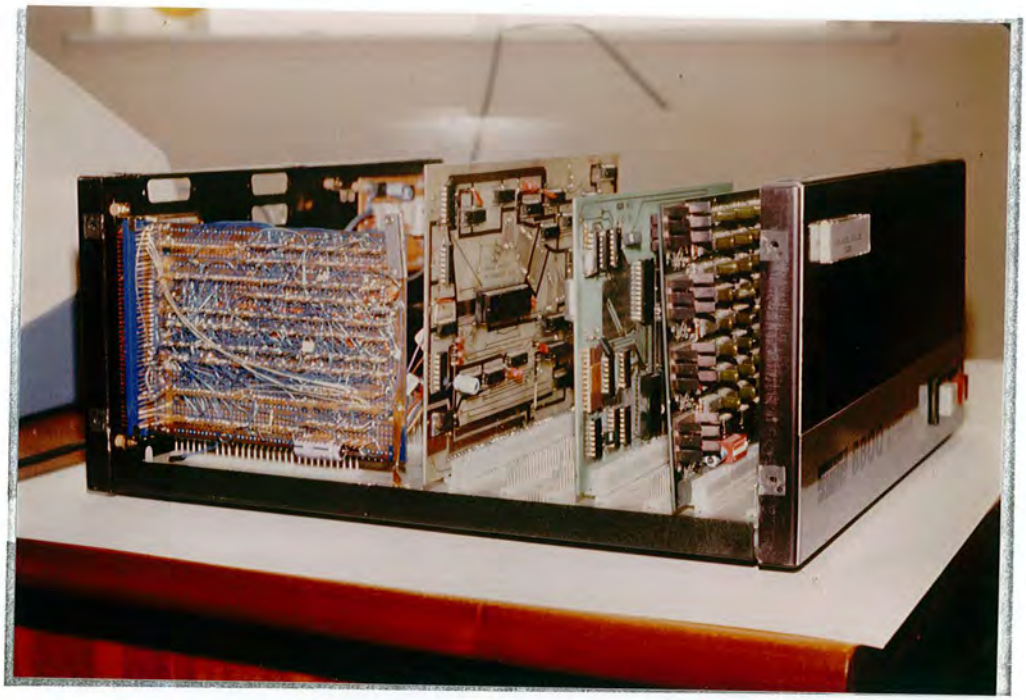


FIGURE 7.7(a). TRANSMITTER HARDWARE (SIDE).

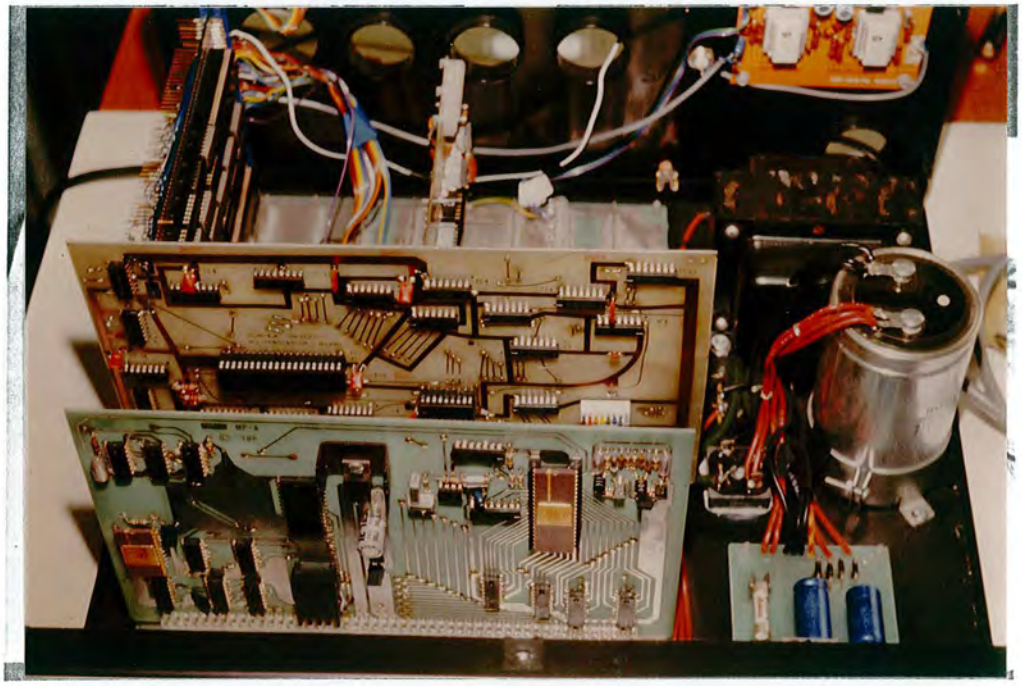


FIGURE 7.7(b). TRANSMITTER HARDWARE (FRONT).

7.3.5 Synchronisation patterns

In order that the receiver may gain element synchronisation, a sequence of phase reversals is transmitted on alternate subchannel slots prior to the frame synchronisation sequence. The phases are equal for each subcarrier during a signal element. The phase reversal sequence is sent over each transmitted subchannel frequency. The element synchronisation pattern is followed by a frame synchronisation pattern, comprising a 31-bit m-sequence pattern. This pattern is transmitted in parallel on all subchannel frequencies. The receiver must attempt to correlate the received synchronisation pattern with the stored sequence.

7.3.6 Construction

The basis for the construction of the transmitter system was the SWTP 6800 development system. The monitor PROM from the "motherboard" of this system was modified to allow insertion of a 2 kbyte EPROM containing the transmitter software. A slave processor unit (see chapter 4) was also mounted onto the motherboard and was interfaced to the modulator wirewrap circuit board via a length of ribbon cable. The A-D conversion system (used to sample the receiver audio output) was constructed on a wirewrap board having a 30-way edge connector to allow insertion onto the I/O bus on the motherboard. A serial interface board was also mounted on this bus, as was an additional board containing a PIA and two reed relays to control (a) enabling of the D-A output signal and (b) the Tx/Rx relay on the HF transceiver. The analogue filter (see chapter 2) was constructed using copper stripboard and later was mounted on the back panel of the system. Photographs of the transmitter system hardware are shown in

figures 7.7 (a) and (b).

7.3.7 Transmitter testing

Because the transmitter is entirely software controlled, it is possible to implement any of the three digital modulation schemes (ASK, PSK or FSK), using up to 8 subcarrier frequencies, by simple software modifications.

To test the transmitter operation, the software was initially set up to generate a phase reversal sequence on a single frequency of 600 Hz. The photograph of figure 7.8 shows the transmitter output signal on the upper oscilloscope trace and the modulator counter reset signal on the lower trace. It can be seen that one signal element comprises 8 cycles of the subcarrier, as expected, and that the counter reset signal appears at the end of a signal element, indicating to the slave processor (via the interrupt routine) that the element is complete. The CCD spectral evaluation module (see chapter 2) was used to display the power spectrum of the voiceband signal, the result of which is shown in figure 7.9. A single peak can be seen in the signal band output, and also in the CCD Nyquist band region.

The software was then configured to permit transmission of sequences using 2,4, and 8 subcarriers. The power spectra obtained from these signals is shown in figures 7.10 (a), (b) and (c) respectively. For the latter case it can be seen that the envelope of the spectrum is not flat and that the transmitted Nyquist band is evident, in addition to the required signal band. These problems were later overcome by improved filtering of the

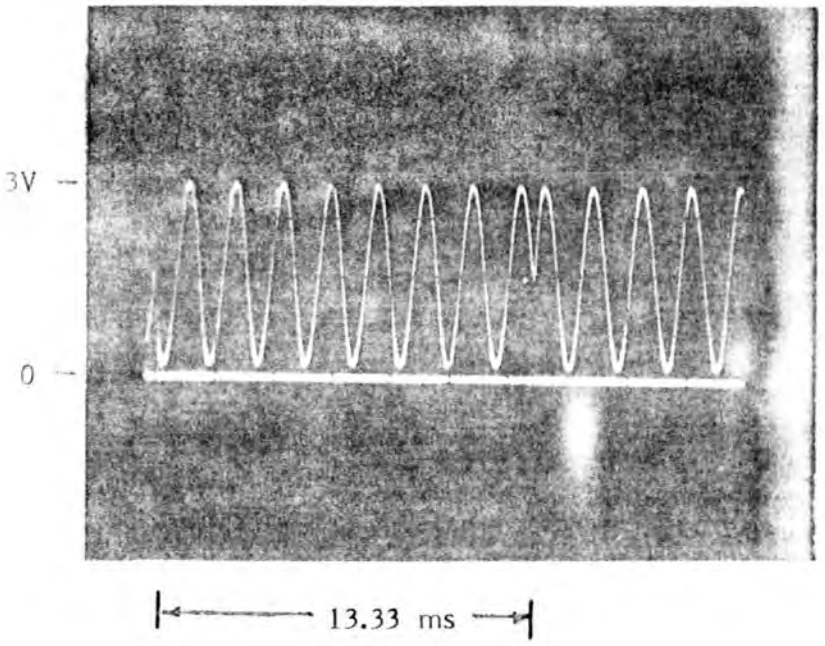


FIGURE 7.8. SINGLE SUBCARRIER PHASE REVERSAL SEQUENCE

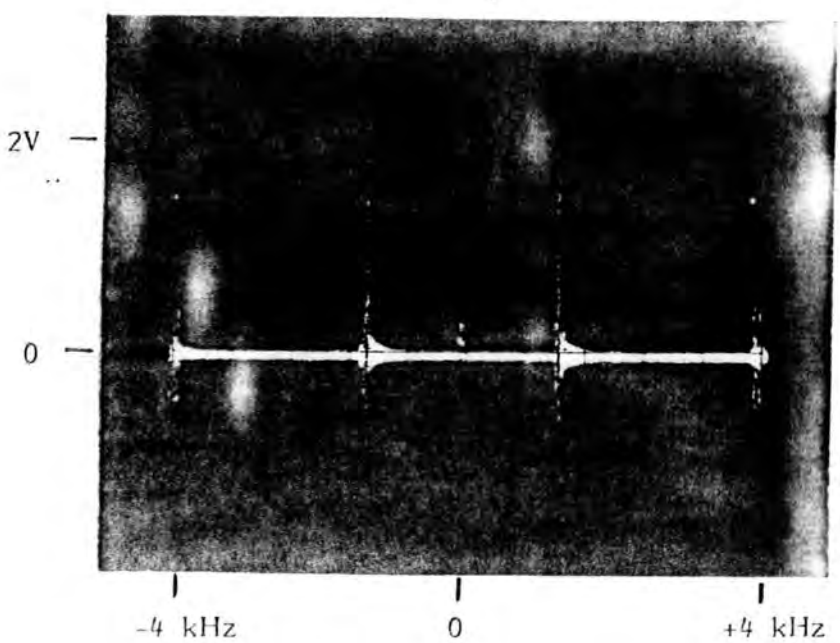


FIGURE 7.9. SINGLE SUBCARRIER POWER SPECTRUM.

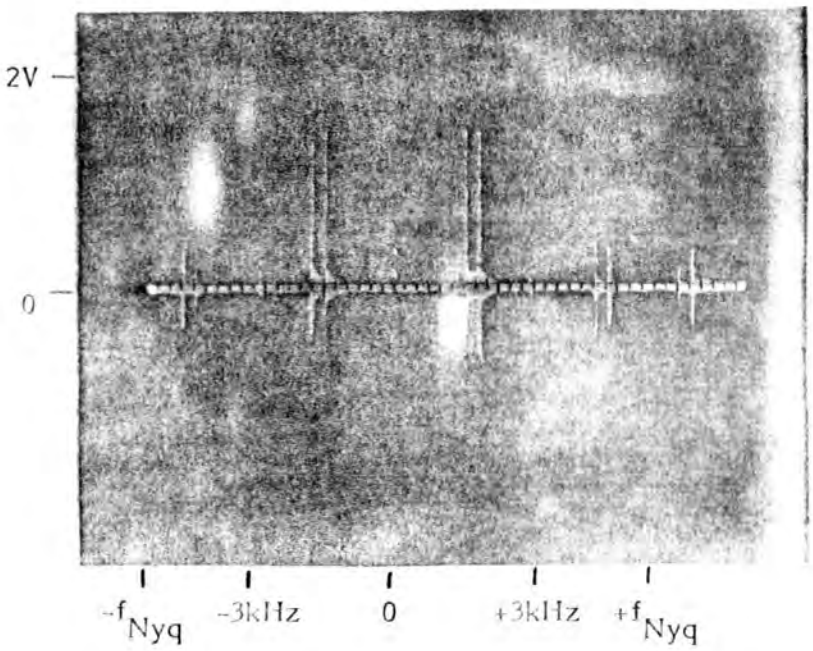


FIGURE 7.10(a). 2 SUBCARRIER POWER SPECTRUM

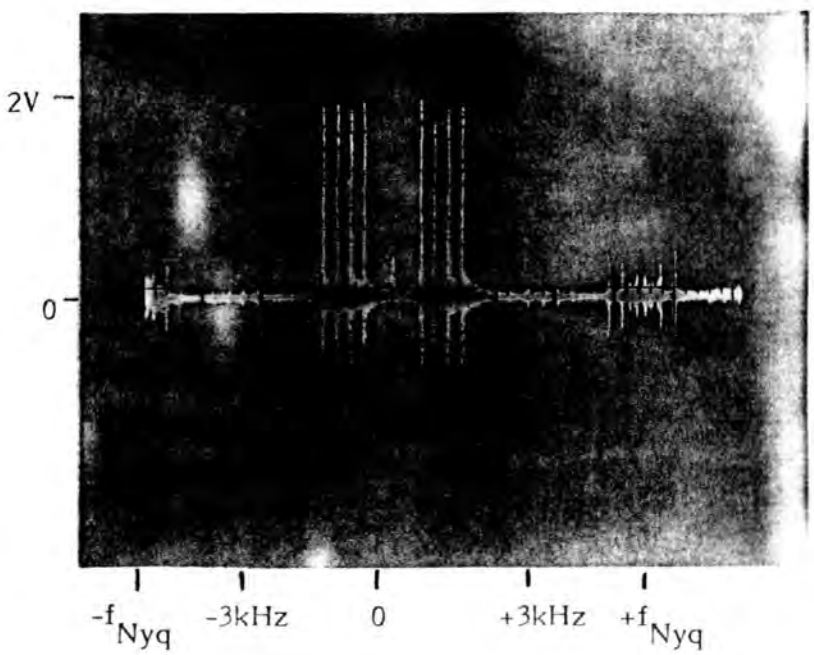


FIGURE 7.10(b). 4 SUBCARRIER POWER SPECTRUM.

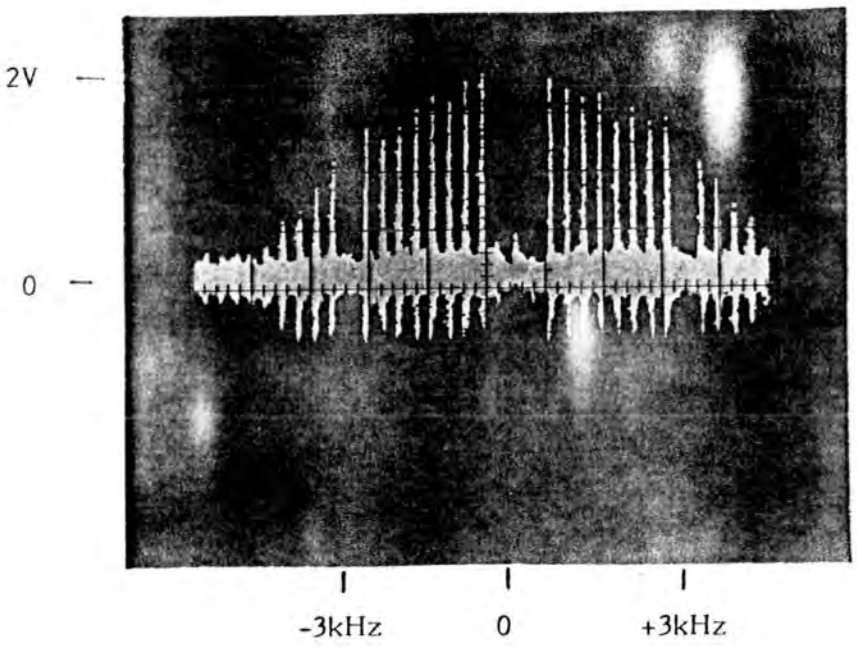


FIGURE 7.10(c). 8 SUBCARRIER POWER SPECTRUM.

D-A converter output signal using the 5-pole filter described in chapter 2, which provided a flat response over the voice channel and a steep rolloff above the 3 kHz band edge.

Although some difficulties were experienced with the receiver demodulation (to be discussed later), it was possible to test the effectiveness of the frequency agility of the transmitter over an HF link. An SSB HF transceiver was available (Collins, type KWT-6), which was installed at the Leicester university field site at Oadby, near Leicester. The antenna socket was connected to an east-west orientated inverted-V half-wave dipole (cut to resonate at 4.7925 MHz) with the apex mounted at a height of approximately 30 ft above ground. An HF receiver (RF Comm. Inc., model RF-505A) was located at Durham; the antenna socket was connected to an east-west orientated half-wave dipole mounted at approximately 100 ft above ground.

The transmitter computing equipment was connected to the HF transceiver as follows. The Tx/Rx reed relay in the microprocessor system was wired to the Tx/Rx switch in the transceiver. The 'transmit' or 'receive' modes could then be controlled automatically. The audio output of the transceiver was connected to the analogue input of the A/D converter. Some difficulty was experienced at first, as the audio gain in the receiver section of the transceiver tended to decrease during the first hour after switching on, after which time the gain remained stable. The gains were therefore set up after the initial "warm-up" period. It was also found necessary to introduce a 0.5s delay after switching from "receive" to "transmit", to allow the

ansceiver internal relays to "make" correctly before commencing transmission.

The spectrum of the received signal at Durham was monitored using the HF spectrogram described in chapter 3. Following this, the "quietest" subchannels at the receiving station were noted during intervals of no transmission. During this time, the transmitter itself was estimating the optimum subchannel slots for subsequent transmission. By observing the radiated spectrum from the transmitter during the next message (ie. by determining the subchannel slots occupied by the signal), it was possible to compare the prediction at the receiving station with that at the transmitting station. Qualitative observations indicated that the prediction at the transmitting site was generally in agreement with that at the receiver.

Observations indicated that a much greater level of agility occurred during the evening, when the interference level was higher, than during the early afternoon (when no detectable interference was observed). On no occasion were more than 4 subchannels reallocated after each message block (transmission time of 3.4 mins), the most frequent number being 2. However, during noisy channel conditions, a reallocation was made on approximately 1 out of every 2 occasions. During midday, little interference was observed, and the frequency allocation remained fixed for more than one hour. The subchannel allocation was observed to coincide with the optimum, as noted at the receiver, on approximately 90% of occasions, indicating that interference observations made at the transmitter site are usually coincident

with those made at the receiver.

7.4 Receiver philosophy

Conventional parallel data modems use narrow-bandpass filter banks to separate out the subcarrier frequencies on reception. This technique suffers from two disadvantages. The cost of constructing a bank of such filters is extremely high and, in a frequency-agile environment, the centre frequencies of the filters must be made adaptive or extra filters must be added, both of which still further increase the cost.

It is possible to use digital techniques to perform matched-filter detection of the received multi-subchannel signal. The Discrete Fourier Transform (DFT) of a finite set of samples may be evaluated and, if it is ensured that the samples all pertain to one signal element, the response of the DFT frequency slots may be arranged to match the spectrum of the transmitted signal. If phase modulation is used, then the Fourier coefficients corresponding to a matching frequency slot (or "bin") will allow determination of the phase for the received signal element. That is, if at the end of a received signal element, the DFT of the sampled version of that element is evaluated, the DFT slots corresponding to the transmitted frequencies may be used to extract the phase for each of the subcarriers.

The functions of the demodulator are twofold: (i) to ensure that all the samples for the DFT calculation pertain to a unique signal element and (ii) to determine the phase from the computed Fourier coefficients. The first criterion may be ensured by observing the variation of the phasor magnitude (for a matched

frequency slot) as the DFT is computed for successive sets of samples over a sequence of phase reversals. The magnitude of the phasor as a function of time is a triangular wave as shown in figure 7.11. The peaks of this waveform then indicate the correct synchronising instants. The second function may be obtained by observing the signs of the real and imaginary coefficients, once synchronisation has been achieved. If the phases are permitted to take on 4 possible values, as in figure 7.3, then, for example, a positive real component indicates that the phasor is located in the right-hand half of the complex (z) plane, and a positive imaginary component indicates a position in the upper half of the plane. This determines a differential phase of $\pi/4$, and the data may be obtained by finding the dibit corresponding to the phase difference between this phase and the phase determined for the preceding signal element (because of the differential PSK modulation scheme).

Once element synchronisation has been achieved, it is a fairly simple matter to obtain frame synchronisation by correlating the demodulated data with a start-of-message synchronisation pattern. This pattern must exhibit good correlation properties when preceded by a sequence of keying inversions. In other words, the correlation coefficient should exhibit a large peak at the synchronising instant and a small amplitude at other instants. A 31-bit m-sequence is known to yield good correlation properties and was chosen as the frame synchronising sequence.

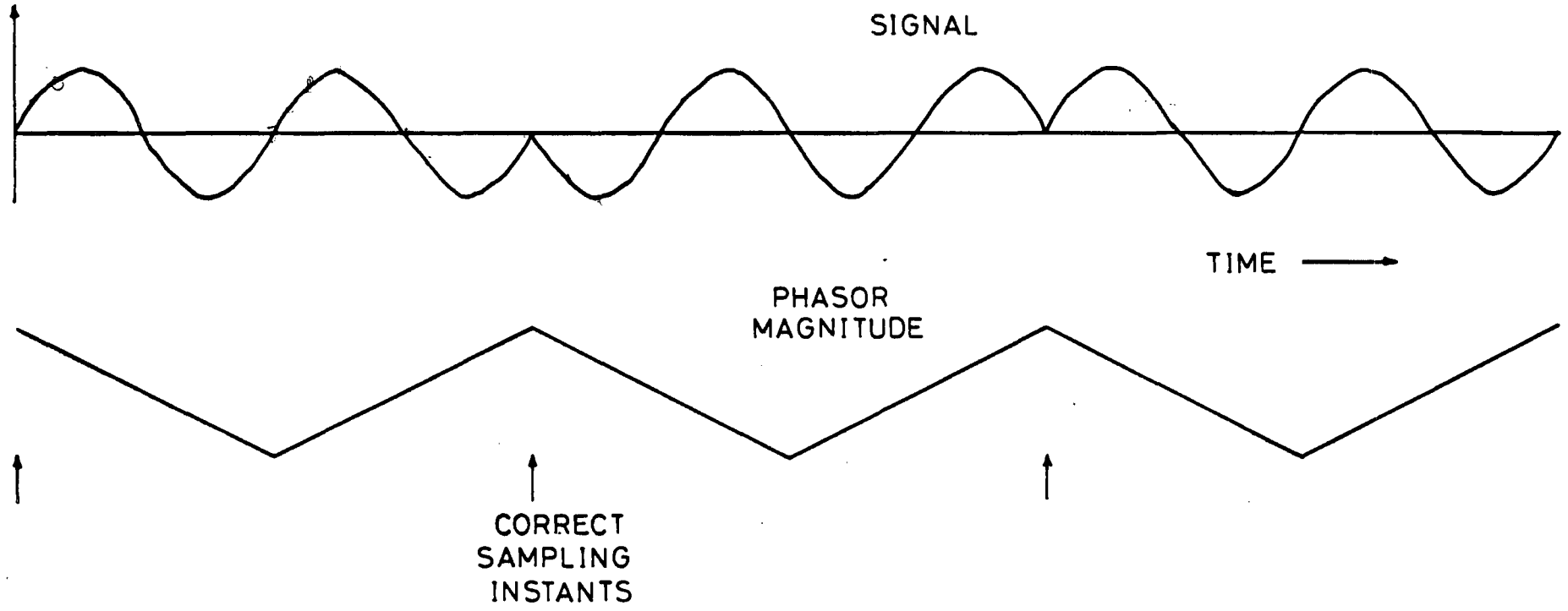


FIGURE 7.11. PHASOR MAGNITUDE VARIATION OVER PHASE REVERSAL SEQUENCE.

In addition to performing the demodulation and synchronisation processes, the receiver must be capable of de-interleaving and decoding the demodulated data. The implementation of these operations has been described in chapter 5.

7.5 Receiver implementation

The computational requirements in the receiver system for the data modem were considerably greater than for the transmitter. The major problem was the demodulation process for which it was necessary to compute the DFT of samples of successive signal elements. A frequency resolution of 150 Hz was required because the available subchannel slots were integer multiples of this frequency. Since the highest permitted subchannel frequency was 2700 Hz it was necessary, at least (by Nyquist's sampling theorem), to compute a transform of length (2700/150)x2 = 36 points. A decimation in time FFT algorithm is suitable for evaluation of transforms of length 2^m (m integer), therefore m was required to be at least 6. The required sampling rate is therefore 9.6 kHz and the resulting DFT slots are integer multiples of 150 Hz from 0 to 4.8 kHz. (The slots above 2.70 kHz are therefore redundant.) It has been shown in chapter 2 of this thesis that the computation time required for a microprocessor implementation of a transform of this length, even using hardware multiplication, far exceeds the signal element duration of 13.33 ms. Consideration was therefore given to a hardware implementation of the DFT using equipment discussed in chapter 2.

A Charge Coupled Device (CCD) was available which, with additional circuitry, could compute a fixed length (512 point) DFT in 5.12 ms or more, depending on the clock rate. The device contains two 512-stage MOS "bucket-brigade" devices which are used to implement four transversal filters using a split-electrode weighting technique. The filters are are used in a "chirp-z" implementation of the DFT algorithm and the device is supplied with circuitry which allows the power spectrum of an analog input waveform to be evaluated. It is necessary to include additional circuitry if, as in this case, the Fourier coefficients are required. The device, its operation, and the design and construction of a module suitable for extracting the complex coefficients from the resulting transform, have been discussed in chapter 2. It has also been shown that the phase of an input signal may subsequently be determined. Reference should be made to this chapter in the subsequent discussion. A receiver design was attempted which utilised the device and its associated circuitry to perform the demodulation process. However, in the course of experimentation, several difficulties became apparent.

At a clock rate of 38.4 kHz, it is possible to acquire 512 samples in precisely the baud time, ie. 13.33 ms. At this sampling rate the frequency domain resolution is 75 Hz. Adjacent subchannel frequency slots are therefore separated by 1 frequency bin and this scheme appears to present a possible solution to the demodulation problem. However, because of the weighting applied to the split electrodes in the CCD filters, a time window is applied to the incoming signal which effectively spreads the power in each frequency bin over several adjacent bins. There is

therefore considerable overlapping between adjacent frequency slots, rendering it impossible to determine the Fourier coefficients using the chosen sampling frequency. It is possible to increase the resolution by decreasing the sampling frequency. However, to ensure that all samples pertain to a single element, it is then necessary to reduce the transform length, which in this case is impossible as the length is fixed by hardware.

To overcome the problem of overlap of nearby frequency bins, the possibility of spreading the input signal spectrum over the frequency range of the CCD was considered. A greater separation between subchannel frequency bins could then be achieved. The CCD operates in a serial fashion; one sample in the time domain is clocked into the device as one sample in the frequency domain is clocked out. To evaluate the DFT of a set of 512 samples of an input signal, it is necessary to enter a replica of the input signal into the device as the frequency domain samples are clocked out, if the true spectrum is to be obtained. To spread the spectrum of the input signal from the HF receiver over the frequency range of the CCD it was necessary to (a) sample the input signal and store the resulting samples, (b) enter the samples into the CCD, then replicate the samples until the CCD was full, (c) obtain the spectrum from the CCD output while entering further replicas of the input signal. It was calculated that, even if the samples were entered into the device at the specified maximum rate of 100 kHz (using a direct memory access (DMA) arrangement), the total time required for all operations was in the order of 20 ms, which was greater than the signal element

duration. These problems therefore precluded the use of the device for the demodulation process. It is envisaged, however, that the advent of new bit-slice microprocessors will permit the computation of the DFT in the required time.

7.6 Phase detection

Some experimentation was carried out into phase demodulation of a single tone carrier using the technique of maximising the phasor magnitude, mentioned in section 7.4. Details of this work are now descibed.

An experiment was set up to demodulate a PSK modulated carrier using software. The transmitter routines were modified to provide, at the transmitter output, a sequence of phase reversals on a 1200 Hz carrier. The signal was sampled by the receiver at a rate of 4800 Hz, resulting in 64 samples for each signal element. The 64 samples were reduced to 32 samples by averaging samples x(n), x(n+32), and a 32-point DFT was computed for each element, as described below. The bandwidth of each of the DFT slots in this case is 150 Hz, which matches the spectrum of the transmitted signal. If a 32-point DFT is computed on 32 successive samples, the received signal will be contained within the DFT slot at k=8.

For the 32 point DFT:

$$F(k) = \sum_{n=0}^{31} f(n)\, e^{-j2\pi kn/32} \qquad k = 0,1, \ldots 31.$$

and

$$F(8) = \sum_{n=0}^{31} f(n)\, e^{-j\pi n/2}$$

$$= \sum_{r=0}^{7} f(4r) - \sum_{r=0}^{7} f(4r+2) + j\left[\sum_{r=0}^{7} f(4r+3) - \sum_{r=0}^{7} f(4r+1)\right]$$

The computation of the DFT slot at k=8 therefore requires 16 additions and 16 subtractions. No multiplications are needed. The magnitude of the function is:

$$|F| = \sqrt{\left[\sum_{r=0}^{7} f(4r) - \sum_{r=0}^{7} f(4r+2)\right]^2 + \left[\sum_{r=0}^{7} f(4r+3) - \sum_{r=0}^{7} f(4r+1)\right]^2}$$

The receiver system for the tests employed 2 slave processors. The first was used simply to interrupt the second at intervals of 13.33 ms, ie. at the signal element duration. The received analog waveform was sampled by the second slave processor using an 8-bit A-D converter and PIA. Computation of the Fourier coefficients was implemented using the master processor. To locate the correct sampling instants, the magnitude of the current phasor (at k=8) was compared with the magnitude of the previous phasor. If the current phasor magnitude was greater than the previous, the timing was advanced or retarded by appropriately adjusting the counter in the first slave. If the previous adjustment was a retardation, then a further retardation was made,

the aim being to maximise the phasor magnitude. Similarly, if the previous adjustment was an advancement, a further advancement was made. If the current phasor magnitude was less than the previous, the timing was made opposite to the previous adjustment.

Once the correct synchronising instants were located, the phase was determined by observing the signs of the Fourier coefficients. It was necessary to make a trade-off between the maximum time required for synchronisation and the amount of tolerable phase jitter. If the timing step adjustment (for a retardation or an advancement) is large, then the time taken to synchronise will be small (because it will take less time to reach the point where the phasor magnitude is maximised), but the jitter will be large because a continual adjustment is being made around the point of maximum magnitude. For a 4-phase system, the amount of tolerable jitter must be less than one-eighth of the period of the highest frequency subcarrier. In the case of the multi-subchannel modem, the highest frequency subcarrier is 2700 Hz, so the maximum tolerable jitter is 46 μs either side of the correct sampling instants. A more realistic figure in a noisy environment might be 20 μs. If the step length is set to this figure, the worst case synchronisation time must then be (13.33ms/20μs)x13.33ms = 8.88s. This is an excessively long time and will result in a considerable reduction in data throughput owing to the long phase reversal sequence required. It is preferable to begin with a large step length and then to reduce the step length as an improvement is observed. A method that was found to be successful was to double the step length if a

deterioration in sync was observed (up to a maximum of 2.5 ms) and to halve the length if an improvement was noticed (down to a minimum of 20 μs). The jitter is then minimised, and phase lock is achieved in a much shorter time (worst case was observed to be 0.19s).

7.7 Conclusion

This chapter has described the design of an adaptive modem for use over HF radio channels which are subject to multipath distortion and noise effects. It has been shown that several modem techniques (TDFM, FEC and frequency agility) may be combined in one system which may be implemented at very low cost by employing nearly-all digital techniques. The use of microprocessors in such designs allows major system changes (such as a change in the modulation or the coding scheme) to be effected by simple software modifications. The use of VLSI technology also allows the physical size of the system to be kept to a minimum.

The transmitter has been described in detail and has been shown to operate successfully over a real HF link. An novel aspect of the transmitter system is the slave processor/ shift register hardware used to generate the modulated signal. However, some difficulties were encountered with the receiver implementation which have been discussed in section 7.5.

Many designs for HF modems have been described in the literature, some of which have been mentioned at the beginning of this chapter and in other areas of this thesis. However, most are extremely costly and non-adaptive, and for these reasons have not presented an economically viable alternative to satellites for

long-distance communications. This chapter has discussed the design of an economical adaptive modem based on discrete signal processing techniques which attempts to overcome many of the problems encountered with previous systems.

CHAPTER 8 Error patterns & coding performance

8.1 Introduction

Radio signals propagated via single or multiple reflections from the ionosphere are frequently subjected to severe levels of amplitude and phase disturbances. The effects on a serial data stream are to cause large numbers of errors which will considerably degrade the fidelity of the received data. The signal distortion arises from (i) intersymbol interference caused by multipath propagation, and (ii) additive noise from natural and man-made sources. The errors which result from these effects are, to a large extent, unpredictable, and will result in a wide range of error rates for a given channel. The error rates may vary by several orders of magnitude even over a relatively short time span. The error distribution is often distinctly non-random in nature, and clusters of errors may occur as a result of noise or fading.

This chapter describes an experiment undertaken to investigate the statistical properties of the error patterns observed over a medium-haul HF radio data link and to assess the performance of the real-time error correction scheme described in chapter 5, with and without interleaving of the codewords. It will be shown that errors occurring over the link are significantly non-random in nature and that bit-interleaved short-length random-error-correcting codes can be effectively used to combat such errors. Results are presented showing the deviations from the theoretical random distributions and the apparent time-varying nature of the error statistics.

Some work on the applications of coding in HF communications systems has been described in the literature (72-75). Much of this, however, has involved recording of the received data; the analysis being performed later using a mainframe computer. The work in this chapter uses microprocessor techniques to perform decoding and error-pattern recording in real time.

8.2 HF Equipment

Experimental tests were carried out over a 250 km south-north path between Leicester and Durham using an HF data link centred on 4.7925 MHz. Permission to use this frequency for data transmission was granted by the Home Office subject to the station callsign (G9BLD) being transmitted in morse code at regular intervals. The HF transmitting equipment comprised a Collins KWT-6 single sideband suppressed carrier transceiver tuned to output 40W PEP into an east-west orientated half-wave dipole antenna situated at a height of approximately 10m above ground level. The transceiver uses valve technology and is constructed in modular form; the synthesised VFO, sideband generator, power amplifier, receiver, and tuning unit are located in separate sections. The audio input frequency response of the transmitter was plotted in the laboratory. The results are shown in figure 8.1 and indicate that the response in reasonably flat within the voice channel spectrum.

The receiving equipment at Durham comprised an inverted-"V" half-wave dipole antenna with the apex situated 25m above ground level (also orientated east-west) feeding into an RF Communications Inc. RF-505A synthesised HF communications

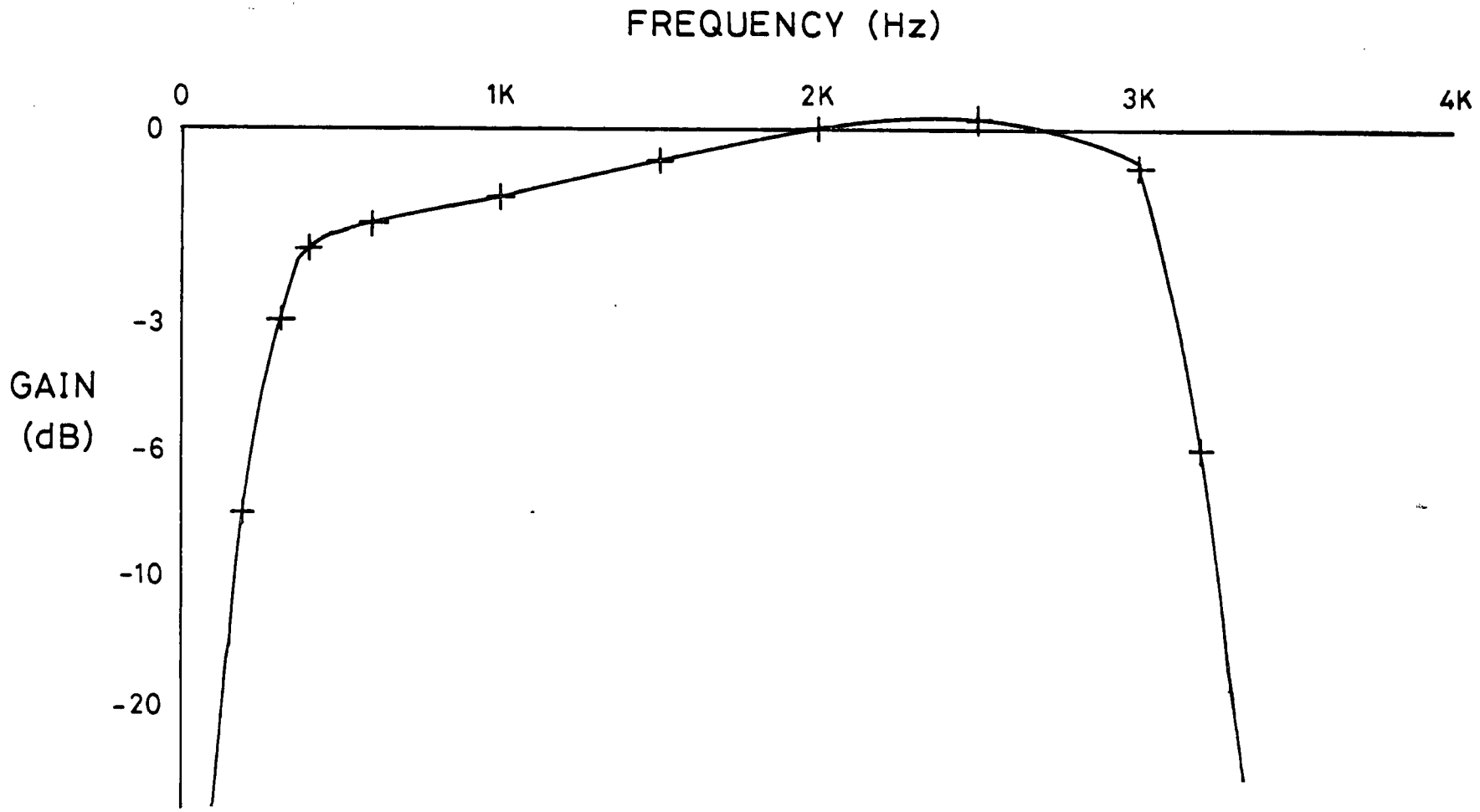


FIGURE 8.1. KWT-6 TX AUDIO INPUT FREQUENCY RESPONSE

receiver. This receiver uses transistor and IC technology and has 3kHz wide independent sideband outputs, as well as CW and AM reception facilities.

8.3 Computing Equipment

All of the computing equipment used for the experiment was based around the Motorola 6800 microprocessor. The transmitting equipment hardware was that described in the previous chapter. It comprised a master CPU board with an EPROM containing the transmitter software, a slave processor unit (described in chapter 4), modulator board and filter unit. The software was configured to produce a binary amplitude modulated audio-frequency carrier at 75 bps at the output of the filter which was then used as input to the HF transmitter. The ASK modulation scheme was chosen to ensure a statistical independence of the errors at the detector. If a differential PSK system was used, the errors would tend to occur in pairs, as the differential detector requires a recovery time of one signal element following the detection of an erroneous bit. The basis of the receiving system was an MSI 6800 microprocessor development system containing two slave processor units, and interfaced to a triple minifloppy disc drive. A real-time clock facility was incorporated within the receiving system which is fully described in section 8.6.7.

The test data format is now described, followed by a more detailed discussion of the transmitter and receiver configurations. Test results are then presented and discussed.

8.4 Data format

Message sequences of 896 bits were used as data for the tests, comprising four consecutive sequences of 32 random 7-bit characters. Two reasons for choosing the 7-bit character format were (a) that each character could conveniently be encoded into a (15,7) codeword and (b) the increasing popularity of the 7-bit ASCII character set as a replacement for the 5-bit BAUDOT code.
The transmission format was as follows:

(1) A sequence of 128 characters, each preceded by a "1" start bit and terminated by a "0" stop bit.

(2) The same sequence, with each character forming the information section of a 15-bit error correcting codeword

(3) The sequence of codewords transmitted in (2), but interleaved to a depth of 16.

Each of the above sequences was preceded by a synchronisation preamble consisting of a series of amplitude inversions, a 15-bit m-sequence and a 7-bit message identification pattern. The m-sequence allowed 3 random errors to occur before synchronisation failure and the identification pattern permitted 1 error before recognition failure.

As mentioned previously, it was necessary to transmit the station callsign in morse code at frequent intervals. The transmitter software included an automatic morse code transmission routine used to transmit the message " DE G9BLD" (the allocated call-sign) at a speed of 12 wpm, approximately every 20 minutes.

8.5 Transmitter

The microprocessor controlled transmitter system hardware has been described in chapter 7. The software may be categorised as follows:

(1) System initialisation & main procedure

(2) Slave processor bootstrapping

(3) Carrier frequency synthesis

(4) Modulation

(5) Data encoding and bit interleaving

(6) Morse code transmission

All software for implementing the above processes was written in M6800 assembly language and assembled into object code format using the co-resident mnemonic assembler. An EPROM programmer, connected to the SS-30 bus of the microprocessor development system, was used to transfer the contents of the object code file onto a single 2 kbyte EPROM. A listing of the transmitter software is shown in the listing in Appendix 2, and a system memory map is shown in figure 8.2. The transmitter routines reside in the EPROM which occupies the top 2 kbytes of the memory address space. The highest two locations contain the program start address (or "reset vector"). The slave processor memory has the (master) address space $C000-$C3FF; the slave control latch is therefore at address $C000, ie. at the bottom of the slave address space. The master system RAM occupies the 4 kbyte space from $0000-$0FFF and a single PIA is located at addresses $8010-$8013. The two least significant bits of the "A" side of this PIA are used to control a pair of reed relays, one of which determines the transceiver operating mode (Tx or Rx); the other

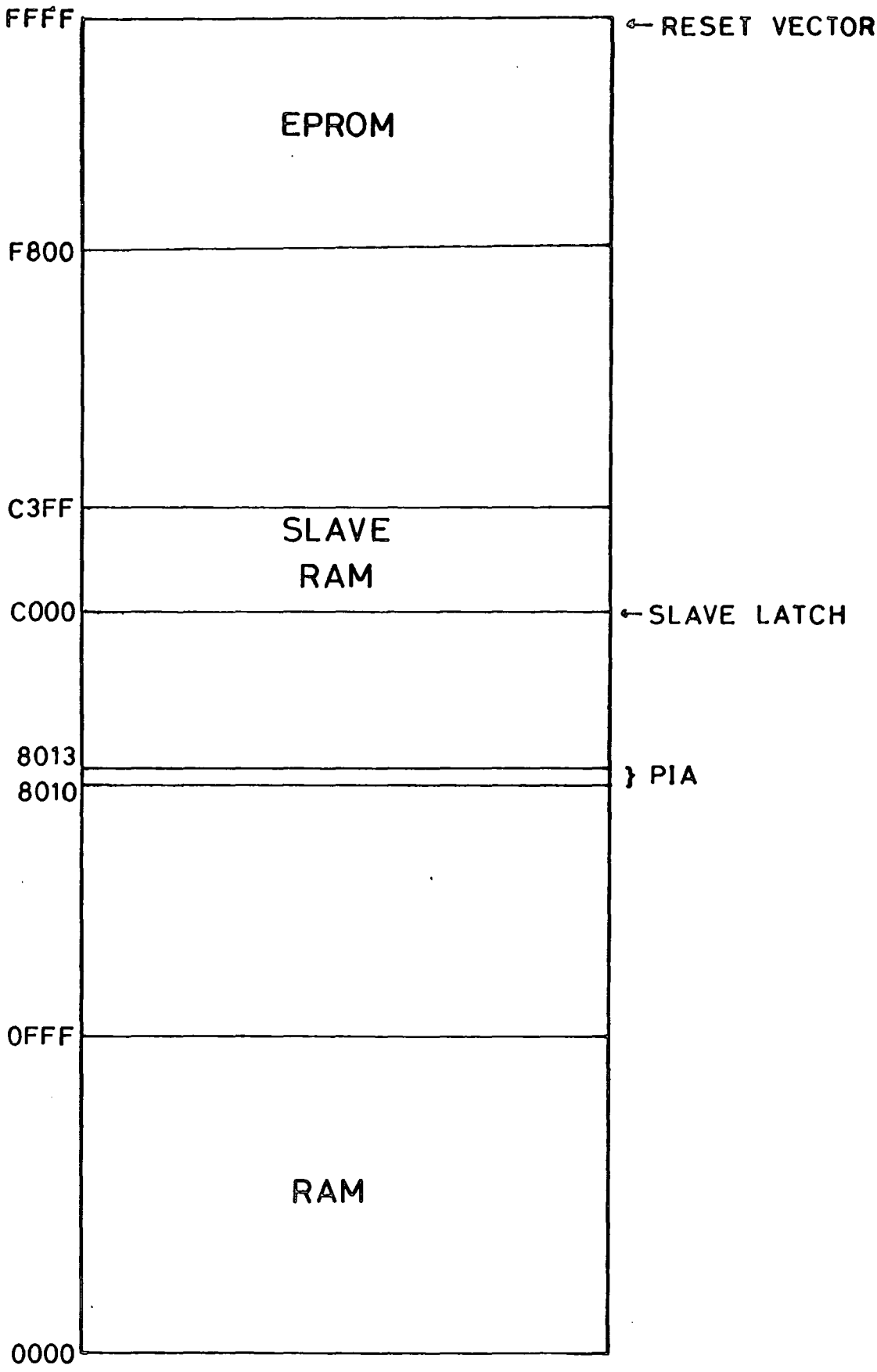


FIGURE 8.2 TX SYSTEM MEMORY MAP

is used to gate the filtered synthesised audio output signal to the transmitter audio input circuitry.

Reference should be made to the transmitter software, listed in Appendix 2. The main procedure begins at the label "INITLSE". The transceiver is initially set to "receive" mode and the filter output is not gated to the transmitter. The slave processor "reset" line is brought low and the slave processor routines are loaded into the slave RAM using the bootstrap loader subroutine. The phase and frequency parameters are then set up in the appropriate tables in slave memory (see chapter 4, section 4.3.6, for details) to initialise all subchannels to a phase of π radians relative to the start of a signal element, and having a frequency of 1500 Hz (centre of the voice channel). The slave processor is then brought into reset by writing a "1" to the slave reset line, the transceiver is switched to "transmit" and the station callsign is transmitted in morse code at a speed of 12 wpm by the morse code transmission subroutine. The first sequence consisting of the uncoded message, followed by the coded message, and finally the coded message with bit interleaving are all transmitted twice. Transmission of the three sequences is repeated in this manner 8 times before the station callsign is transmitted again. 75 bps.

8.5.1. Bootstrap loader

The bootstrap loader routine simply maps the slave processor control program from the EPROM into the slave RAM area. Slave processor execution is suspended during this procedure by holding the slave reset line at logic '0'. The beginning and end

addresses of the area to be mapped are contained in Y,Y+1 and Z,Z+1. The index register points to the start address of the area into which the data is to be mapped.

8.5.2 Carrier frequency synthesis

Synthesis of the AF carrier is performed by the slave processor and the modulation board containing the shift registers and D/A converter. The synthesis of a multitone carrier using this system has been described in chapter 4. For this experiment a single tone was required, and all eight subchannels were set to the same frequency and phase. The first 40 bytes of the slave memory contained (after bootstrapping) the lookup table required for carrier generation, consisting of equally spaced samples of a sinusoid. The slave reset vector is located in the top two bytes of the slave address space, ie. at local (slave) addresses $03FE and $03FF, and the local IRQ vector is located at $03F8 and $03F9. An IRQ is initiated by the modulator circuit immediately following the transmission of samples for a complete signal element.

8.5.3 Modulation

An ASK modulation scheme at 75 bps was used for the experiment. The transmitter software was configured to generate an on-off keyed audio tone at 1500 Hz. Because of the "lookup table" method of carrier synthesis, the phase at the start of a transmitted signal element is always constant. The modulation process is carried out by the subroutine "SNBITS", which transmits N left-justified bits contained in the A accumulator, where the bit count, N, is contained in the B accumulator. If a

bit is "0", the step lengths in the slave processor subchannel table are all set to zero, thereby inhibiting carrier transmission during that element. If the bit is "1" the step lengths are all set to 20 and the element comprises 20 cycles of a 1500 Hz sinusoid.

8.5.4 Morse code transmission routine

This routine is used to transmit the message "DE G9BLD" when called. The output gating relay is keyed according to the bits in a stored sequence which represents the message. The relative timing of the elements comprising the morse characters was arranged to conform to the following internationally accepted schedule (9):

dash	=	3 dots
element space	=	1 dot
letter space	=	3 dots
word space	=	6 dots

The key is initially set to "open" - ie. the carrier is not gated to the transmitter. Bits are read sequentially from the sequence "MMES" and if the bit that is read is a "1" the state of the key is switched; if it is "0" nothing is done. After each bit has been read, and the key switched if necessary, a fixed delay is introduced which governs the overall transmission rate of the morse message. The sequence required to transmit the characters "DE G9BLD" is shown in figure 8.3.

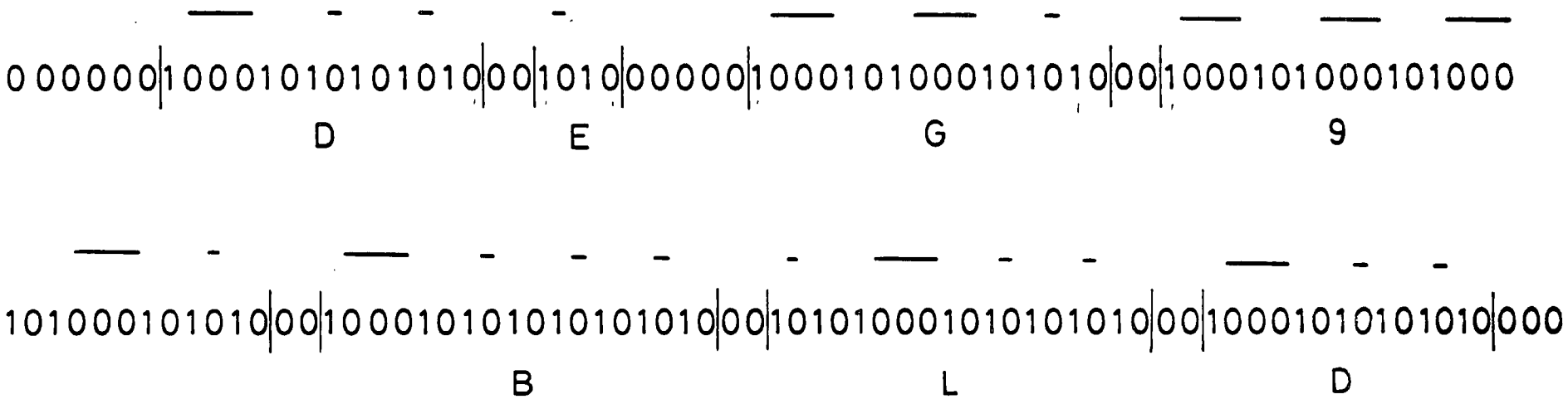


FIGURE 8.4. MORSE CODE TRANSMISSION SEQUENCE

8.6 Receiver

The receiving equipment consisted of the HF radio receiver with dipole antenna, ASK demodulator unit, MSI 6800 development system with 2 slave processor units (see chapter 4), triple floppy disc drives, and printer. The hardware configuration of the system is shown in figure 8.4. One slave processor was required for real-time decoding and de-interleaving of the received data, the other was used as a timer to locate the correct sampling instants on the demodulated signal. The floppy disc drives were used to store statistical information about the received data for later analysis. The receiver functions may be categorised as follows:

(1) demodulation

(2) synchronisation

(3) de-interleaving & decoding

(4) error counting

(5) error pattern recording

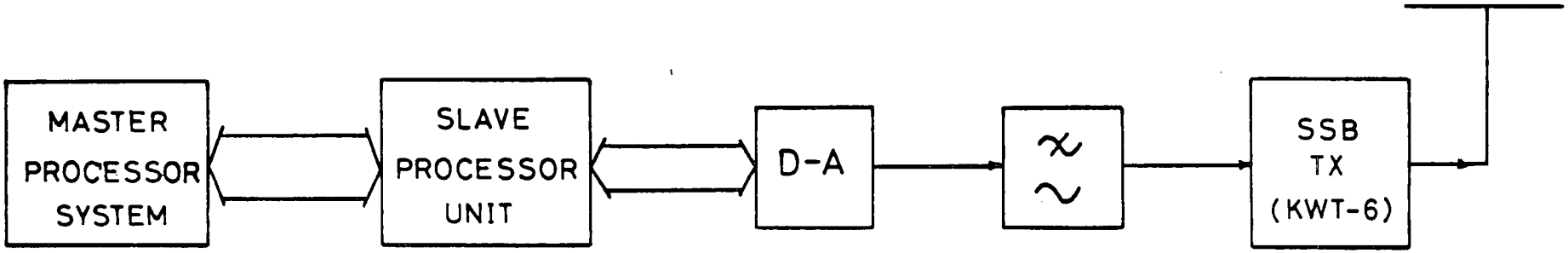
(6) disc management

(7) real-time clock

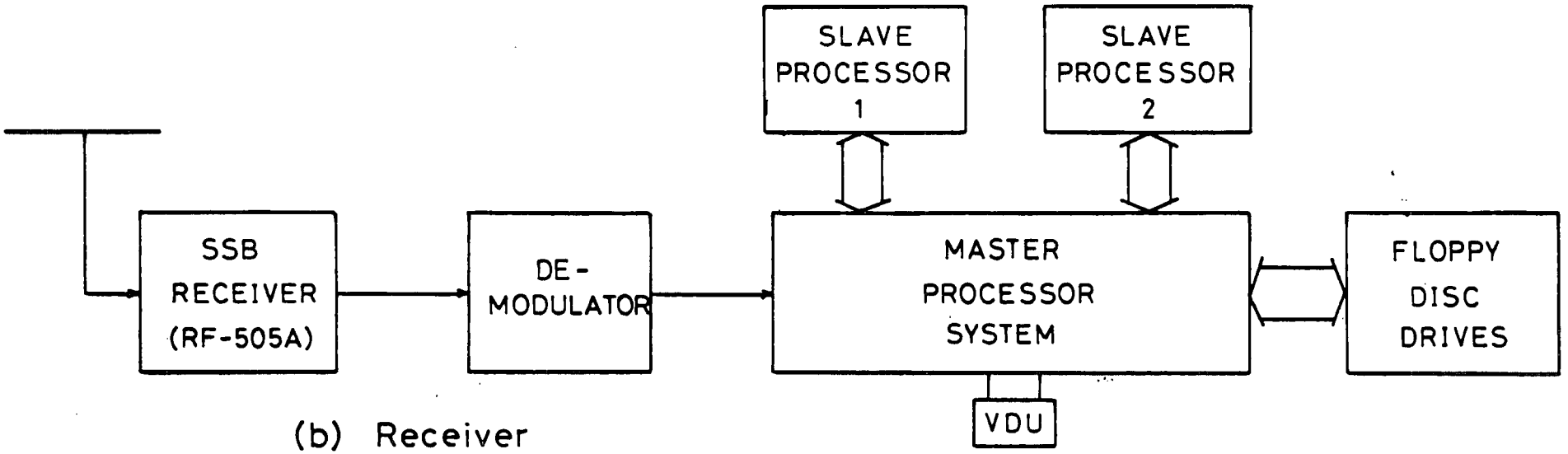
Each function is now described in turn.

8.6.1 Demodulator

The circuit of figure 8.5 was devised to permit incoherent envelope detection of the received signal. The USB audio output from the HF receiver was buffered and half-wave rectified by the diode. The envelope was extracted by low-pass filtering using C and R_1, then buffered and finally converted to a rectangular baseband signal using the Schmitt trigger. The latter was preferred to a straightforward comparator in order to minimise



(a) Transmitter (Leicester)



(b) Receiver

FIGURE 8.3. HARDWARE CONFIGURATION

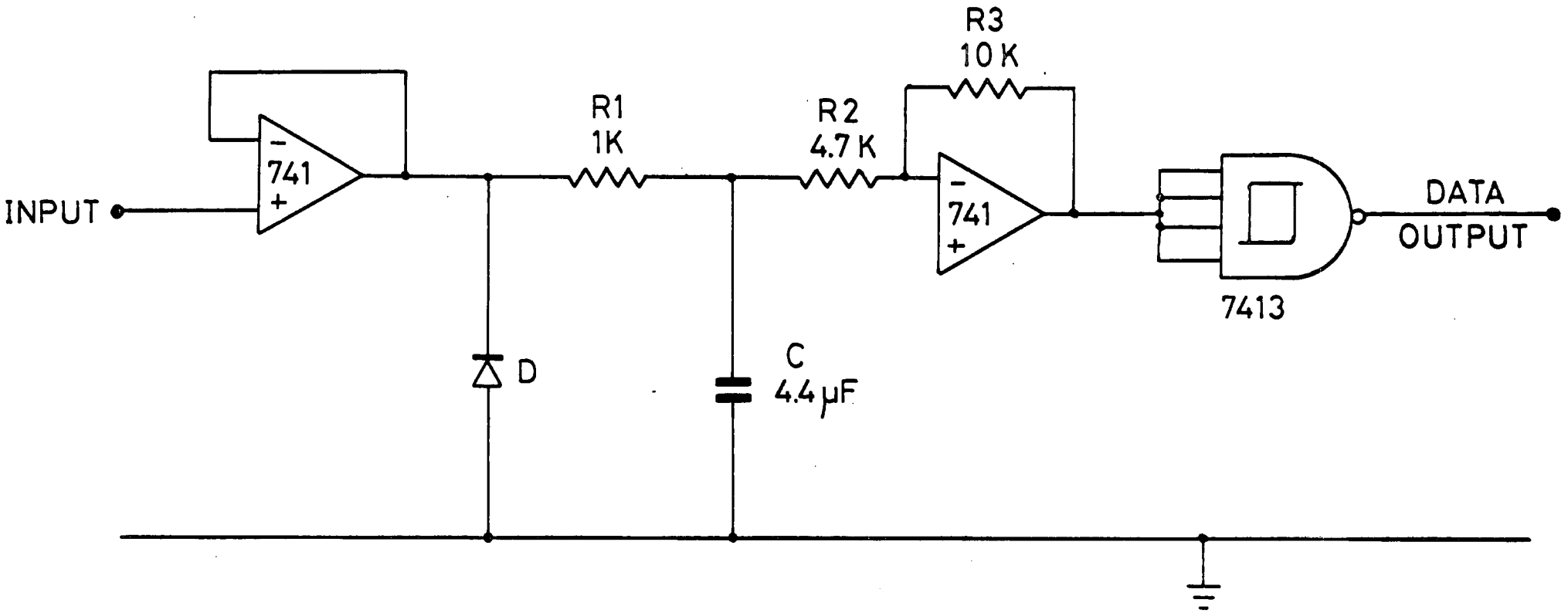


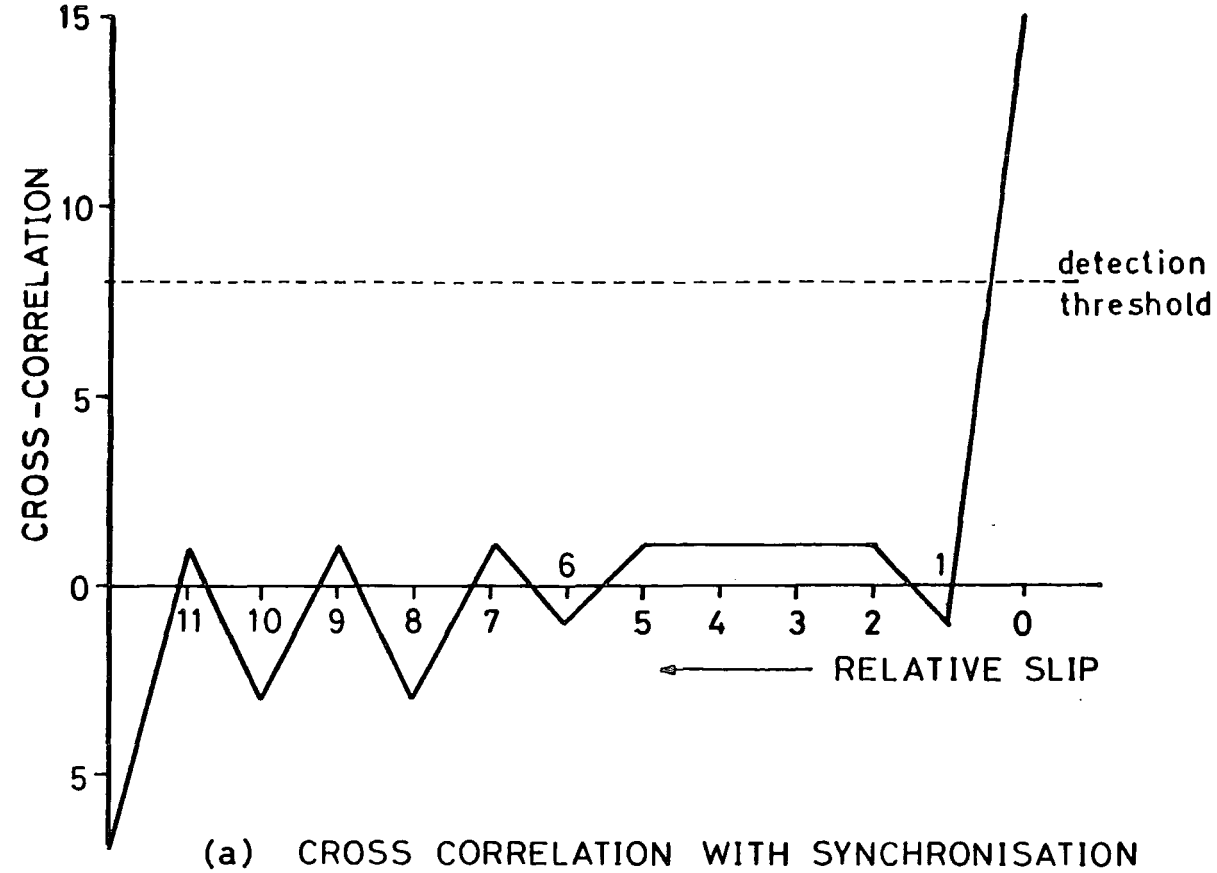
FIGURE 8.5. DEMODULATOR CIRCUIT

noise effects. The output data signal was then fed to the most significant bit of one port of a PIA interfaced to the main processor system.

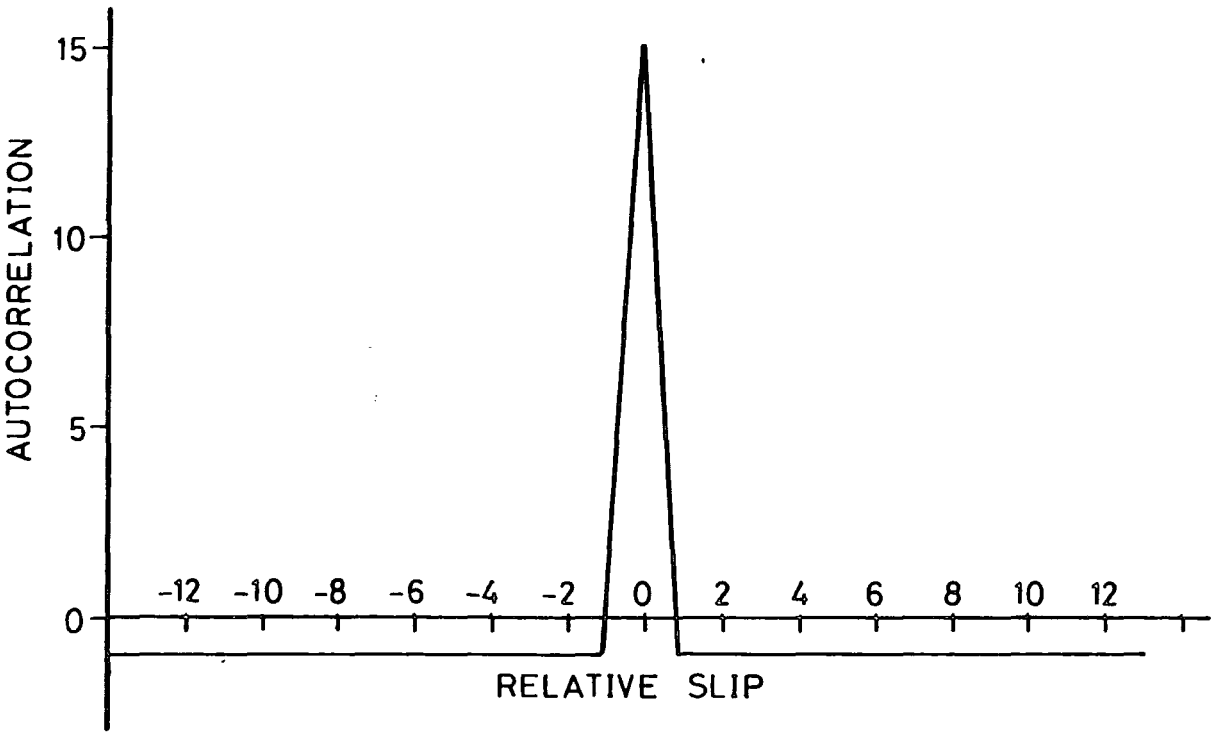
8.6.2 Synchronisation

The preamble sequence transmitted immediately before each message was used to obtain element and frame synchronisation. The purpose of element synchronisation was to ensure that each signal element was sampled as near as possible to its mid point. The first part of the preamble consisted of a sequence of amplitude inversions which was used by the receiver to locate the correct sampling instant to within 2% accuracy. Once synchronised, a software timing loop in the second slave processor unit was used to interrupt the master processor at the centre of each signal element.

Frame synchronisation was achieved using a 15-bit m-sequence which was known to yield good correlation properties when preceded by the amplitude inversion sequence. The cross-correlation between the stored m-sequence and the synchronisation preamble is shown in figure 8.7(a). The autocorrelation property of the same sequence is illustrated in figure 8.7(b). During synchronisation, each received bit is shifted into a 2-byte buffer (SNPAT) and a correlation is performed with the stored sequence (MSEQ) by finding the Hamming weight of the result of an exclusive-OR operation between the low-order 15 bits of SNPAT and the sequence MSEQ. A low weight indicates a good correlation. A correlation factor ≤ 8 indicates detection of the synchronisation pattern. The distance from the detection threshold to the correlation peak is 7



(a) CROSS CORRELATION WITH SYNCHRONISATION PREAMBLE.



(b) AUTOCORRELATION FUNCTION

FIGURE 8.7. m-SEQUENCE PROPERTIES

which allows up to 3 random errors in the sync pattern before the receiver fails to detect it. (This is because a distance of at least 2t+1 is required if t errors are to be corrected.)

The message identification pattern was transmitted immediately following the m-sequence, and consists of one of three codewords from an m-sequence of length 7, the codeword indicating whether the message is uncoded, coded, or coded and interleaved. The distance between different words in this code is 4, which allows correction of a single error.

8.6.3 De-interleaving & decoding

Slave processor 2 was used for real-time de-interleaving and decoding of the received data. Operations by the slave were carried out on one of two pairs of tables containing the data while operations on the other pair were performed by the master. One table in each pair contained data for de-interleaving (if required); the other contained received words for decoding. Two control parameters were passed to the slave from the master indicating (a) which pair of tables were to be operated on (TBFLAG), and (b) whether or not de-interleaving was required. A pictorial representation of the operations is shown in figure 8.8. The de-interleaving process was executed by the subroutine DINTLV and the resulting words were entered into the appropriate decoding table as they were extracted. On completion of this operation, the 16 received words in the decoding table were decoded using the algorithm described in chapter 5. Each decoded word was returned to its corresponding position in the table. A count of the total number of errors corrected during decoding was kept in the

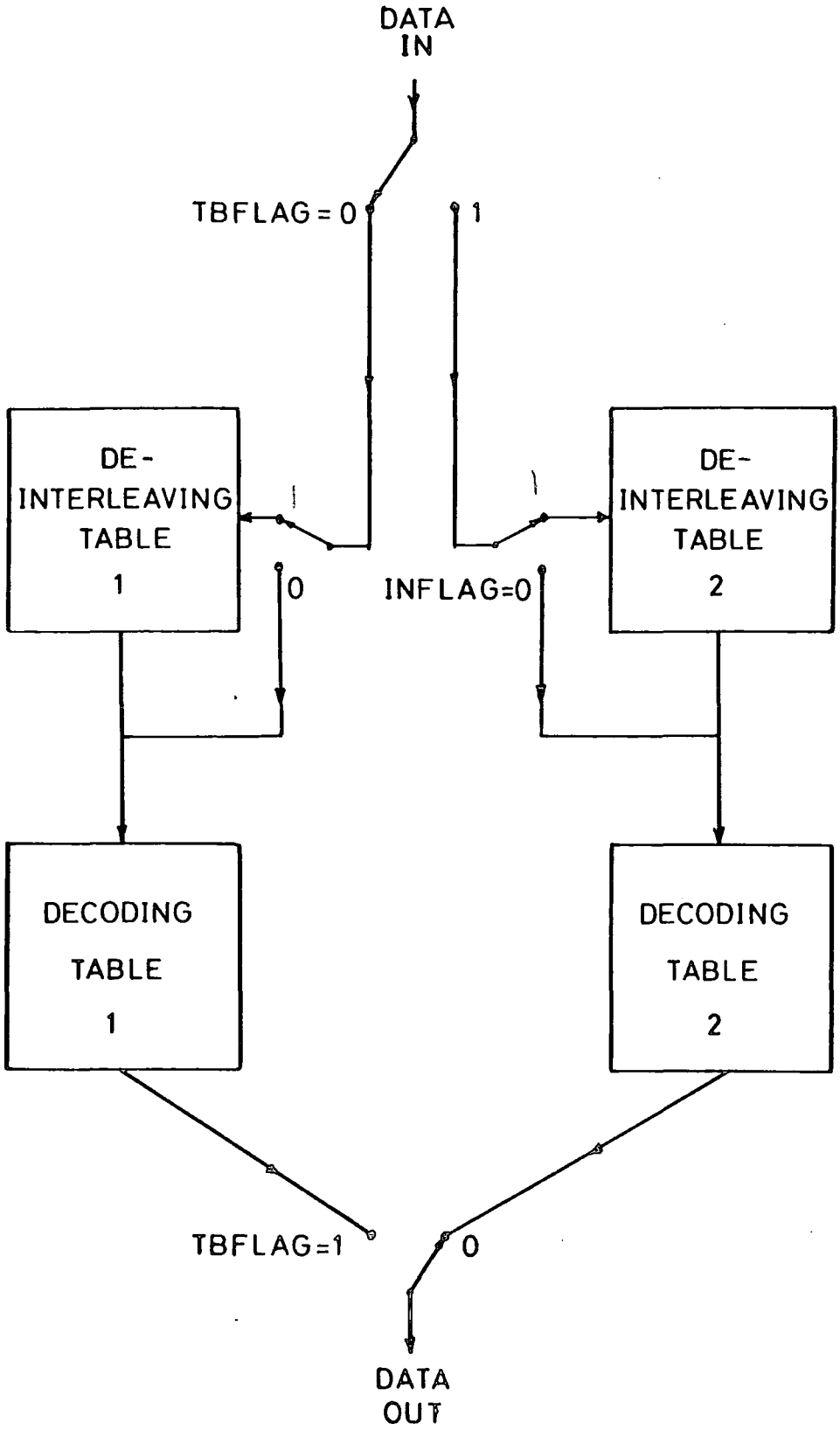


FIGURE 8.8 DIAGRAMMATIC REPRESENTATION OF DE-INTERLEAVING & DECODING PROCEDURES.

two-byte counter "ERRCOR".

While decoding and de-interleaving operations were performed by the slave, the newly received data was entered into the inactive pair of tables in slave RAM. Coded but non-interleaved data was entered directly into the decoding table. When the slave operations were complete, the decoded data was offloaded by the master and the tables were switched to allow the receiving process to continue.

8.6.4 Error counting

The number of errors in a decoded message was determined by cross-correlating the decoded words with a stored table of correct characters. This cross-correlation was performed by implementing an exclusive-OR operation between the decoded word and the corresponding stored character. The least significant 7 bits of the result were then shifted out into the carry flag, in turn, and if the flag was set, the bit was found to be erroneous. The total number of corrected errors was obtained from the slave processor on completion of the decoding procedure.

The total number of errors in the received bit stream (before decoding) was determined by a similar correlation procedure and the statistical properties of the error patterns were recorded as described in the next section. Results of the error counts and the number of errors corrected were stored on disc.

8.6.5 Error pattern recording

As each bit was received a test was made to determine whether or not the bit was erroneous. This was done by correlating the received data with the stored (correct) bit pattern. Three tables of stored patterns were required; one for each of the uncoded, coded and coded and interleaved sequences. A flag (PREV) was used to indicate if the bit immediately preceding the current bit had been received correctly, thereby allowing the number of consecutive erroneous bits to be recorded. A count of the number of consecutive correctly received bits was recorded as a positive number (1 or 2 bytes) on the disc and a count of the number of consecutive errors was recorded as a 2's complement negative number. A complete record of the error pattern information for one received sequence therefore consisted of alternate positive and negative values which could be later retrieved for analysis. A 2-byte "start-of-message" marker ($FFFF) was recorded at the beginning of each data set to distinguish the records. Because it was often necessary to record on disc a large number of bytes for each received sequence, 2 disc drives were used to store the information. When the disc in drive 1 was full, the file (ERRPAT.000) was closed and a new file (with the same name) was opened on drive 2. The file capacity was therefore increased from 80 kbytes to 160 kbytes and allowed a useful quantity of data to be collected ($>$ 24 hours) without exceeding the disc storage capacity.

8.6.6 Disc management

The disc file management software allowed sequential access disc files to be created, written, read and destroyed by the system user and is described in reference (76). Two files were used to store information on the received data; therefore two file control blocks (FCBs) were required. A file on disc drive 0 (called 'ERRDAT.000') was used to record the following parameters in sequential order:

(a) time of day (three 2-digit BCD numbers)

(b) message type (1-uncoded, 2-coded, 3-interleaved)

(c) no. of errors in decoded message (2 bytes)

(d) total no. of errors in received bit stream (2 bytes)

(e) total no. of corrected errors (2 bytes)

A second file (called 'ERRPAT.000') was created, initially on drive 1, to store error pattern data on the incoming bit stream.

8.6.7 Real-time clock

A real time clock was kept within the processor system memory by using the M6800 interrupt capability. A 1Hz timing reference was derived from a digital clock unit installed in the receiver rack to display the current time of day. The 1Hz TTL output from this unit was connected to the CA1 control line of a PIA which was configured to cause an interrupt to be generated on detection of a negative edge on this line. If bit 0 of the control register is set to a '1' and bit 1 is set to a '0', a negative edge on CA1 will cause an interrupt to be generated by setting the IRQA1 flag (bit 7), which causes the IRQ line on the processor to be brought low. The system monitor interrupt routine then causes a jump to the address stored at the lowest two locations in the monitor RAM.

This address is the start of the clock routine. The clock routine updates the time of day which is stored in BCD format in four bytes located on page 0, one each to hold seconds, minutes, hours and days. The penultimate instruction in the clock routine clears the IRQA1 flag by reading peripheral data register A. A 'return from interrupt' instruction then restores the stack and execution of the interrupted program is resumed. The fastest execution time for the clock interrupt routine is 31 machine cycles; the slowest is 84 cycles.

8.7 Data analysis programs

The experimental results were based on data collected during the week of 10/7/80 to 16/7/80. Propagation conditions were found to be similar from day to day. Two programs were written in BASIC to analyse the data. The first was used to obtain the following information from the file ERRDAT.000, averaged over each hour of the observation period:

(1) BER averaged over all received messages.

(2) BER averaged for each of the three received sequences.

(3) BER after decoding for each of the 2 coded sequences.

(4) Proportion of errors corrected for serial and interleaved codeword sequences.

The second program was used to obtain statistical information on the error patterns stored in the file ERRPAT.000. The following information was obtained from this program:

(1) Frequency of occurrence of consecutive errors.

(2) Frequency of occurrence of error free intervals.

(3) Frequency of occurrence of error bursts.

Assembler-written routines were called by the BASIC user function, USR(X), to control the reading of data blocks from the disc files.

8.8 Experimental Results

The results section of this chapter is divided into two subsections:

(1) Statistical analysis of error patterns.

(2) Analysis of coding performance.

The coding performance results will be seen to correlate with the error pattern analysis.

8.8.1 Analysis of error patterns

Distributions of consecutive errors, consecutive error-free intervals, and burst error occurrence were obtained from the recorded results.

For a random error distribution, the theoretical probability, P_e, that n consecutive errors occur when the average probability of a bit error is p, is given by:

$$P_e(n) = p^n(1-p)$$

This function is tabulated in table 8.1 for two chosen values of BER. For a BER of 10^{-2} it can be deduced that 99% of the errors should occur singly; for BER = 2×10^{-2} the figure is 98%.

n	1×10^{-2}	2×10^{-2}
0	9.9×10^{-1}	9.8×10^{-1}
1	9.9×10^{-3}	1.96×10^{-2}
2	9.9×10^{-5}	3.92×10^{-4}
3	9.9×10^{-7}	7.84×10^{-6}
4	9.9×10^{-9}	1.57×10^{-7}

Table 8.1

The error performance of the HF link is plotted in figure 8.9 for a 24 hour observation period. The averaged raw BER (without coding) per hour was observed to vary between 3.2×10^{-3} and 1.94×10^{-2}. Two hourly observation periods were chosen having BERs close to those used to compute the theoretical distributions; these were the periods 2300-0000 hrs. and 1400-1500 hrs. which corresponded to average bit error rates of 0.98×10^{-2} and 1.94×10^{-2} respectively. (figures 8.10(a) and (b)). The results indicated that around 75% of the errors were single, and 15% occurred in pairs. It can be seen that the measured distributions deviated substantially from the theoretical random estimation, but that the measured distributions appeared similar for both values of BER. In other words the consecutive error distribution appeared to be independent of error rate. It was also noted that the distribution appeared to be exponential. A regression analysis showed that a function conforming closely to the measured distribution is given by:

$$f_o(n) = 74\ e^{(-\ 6.2(n-1))}$$

where $f_o(n)$ is the frequency of occurrence (%) of n consecutive errors. The results indicated that the errors were tending to cluster and that the size of the clusters was independent of the average error density.

The cumulative distribution of error-free intervals was plotted for the two hourly periods analysed above and compared with the theoretical random distributions. The theoretical probability, $P_f(n)$, of an n-bit error-free interval assuming random error statistics is given by:

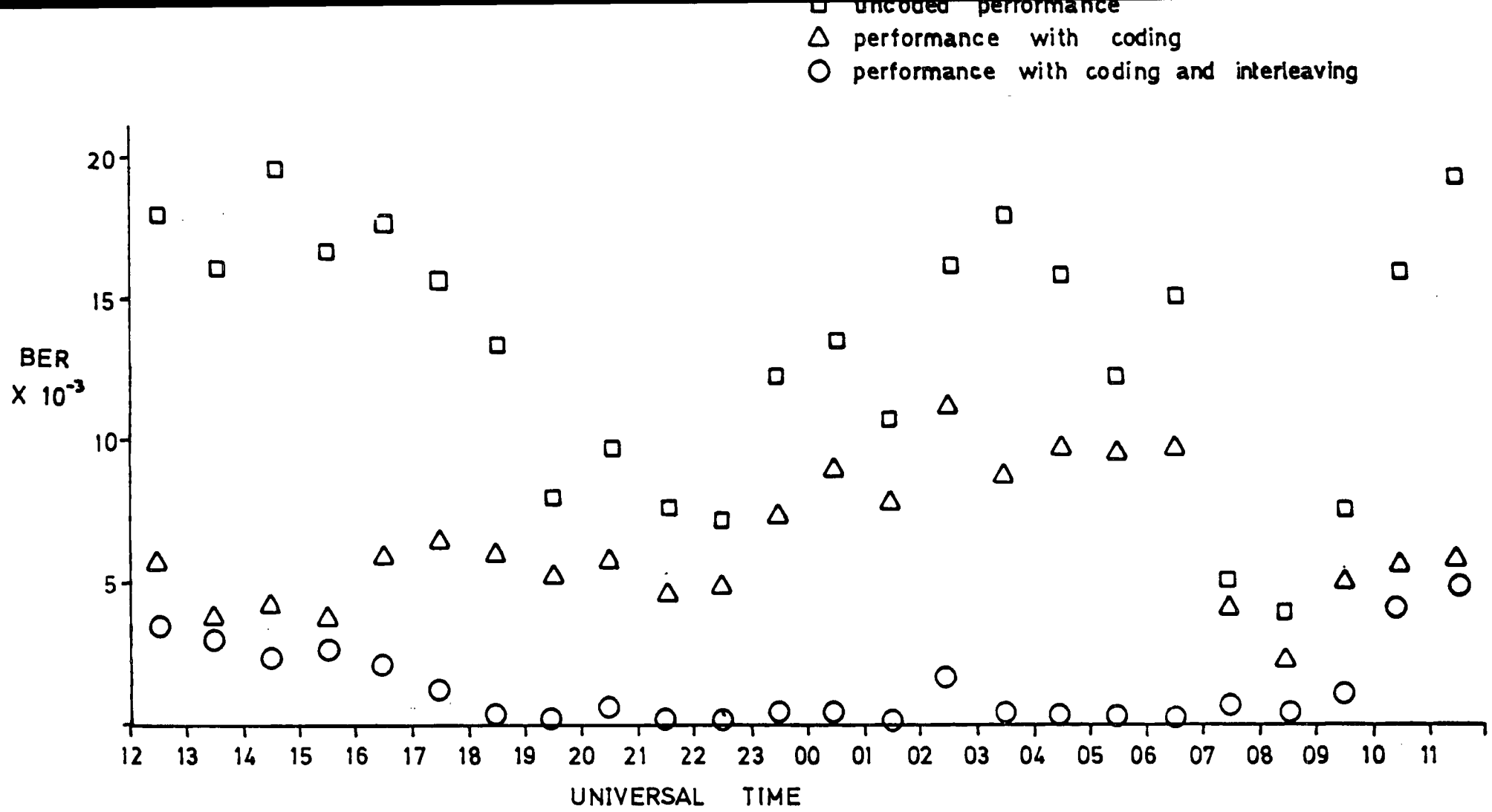


Fig. 8.9. TYPICAL ERROR PERFORMANCE OF HF LINK

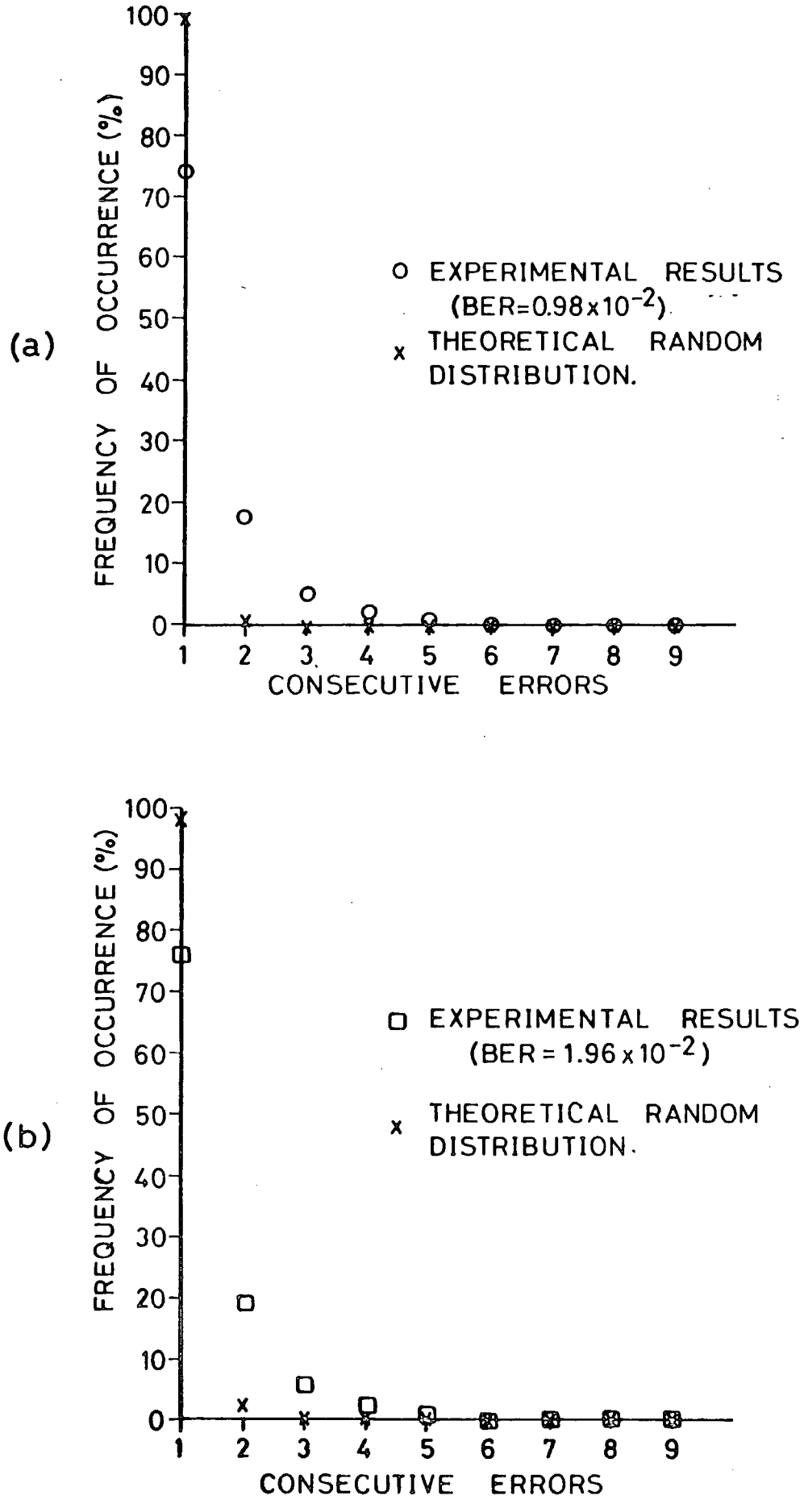


FIGURE 8.10. DISTRIBUTIONS OF CONSECUTIVE ERRORS.

$$P_f(n) = p(1-p)^n$$

where p is the probability of a bit error. Again, the results (figs 8.11 (a) and (b)) indicate substantial deviations from the random distribution, and appear to be independent of error rate. For a BER of 10^{-2}, the random distribution predicts that only 10% of the errors should be separated by a gap of 10 bits or less, whereas the measured figure was 76%.

An error "burst" is defined in a paper by Brayer (77) to be a region of the serial data stream where the following properties hold. A minimum of 2 errors are contained in the region and the minimum density of errors in the region is Δ. The burst error must always begin with a bit in error and end with a bit in error, and must be immediately preceded and followed by an interval in which the density of errors is less than Δ.The burst density criterion Δ in this paper was chosen to be 0.05. This definition of an error burst did not appear to be entirely satisfactory as the following example illustrates.

Suppose that the first 5 and the last 1 of 120 consecutive bits are received erroneously. The error density is therefore 6/120 = 0.05 which is exactly the minimum density required to define a burst. The data would be logged as containing a burst of errors of length 120 even though there is an error-free interval of 114 bits. It would intuitively seem more reasonable to suppose that the errors were distributed as a burst of length 5 followed by a single random error occurring 114 bits later.

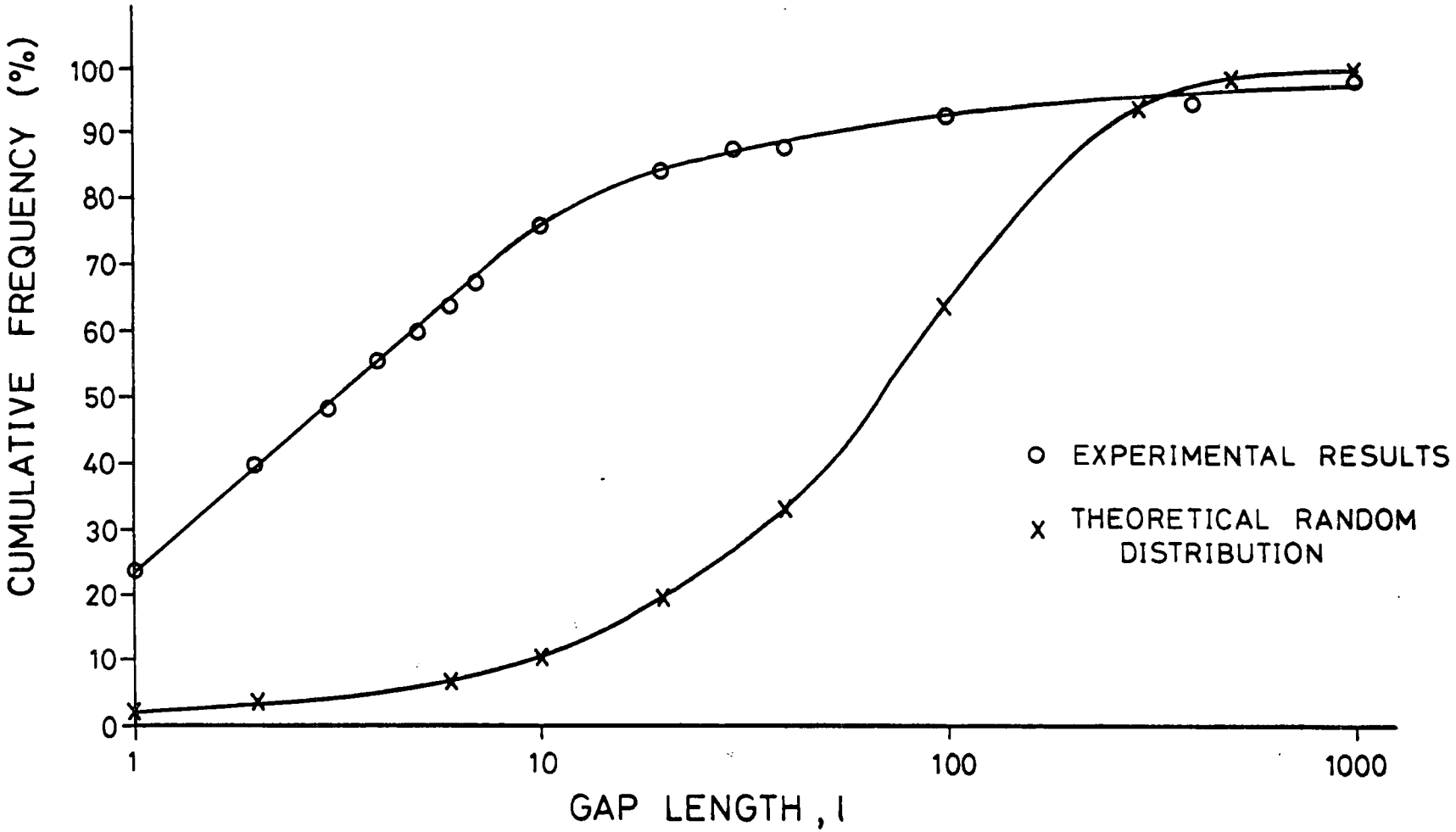


FIGURE 8.11(a). DISTRIBUTION OF ERROR-FREE INTERVALS.
(BER = 0.98×10^{-2})

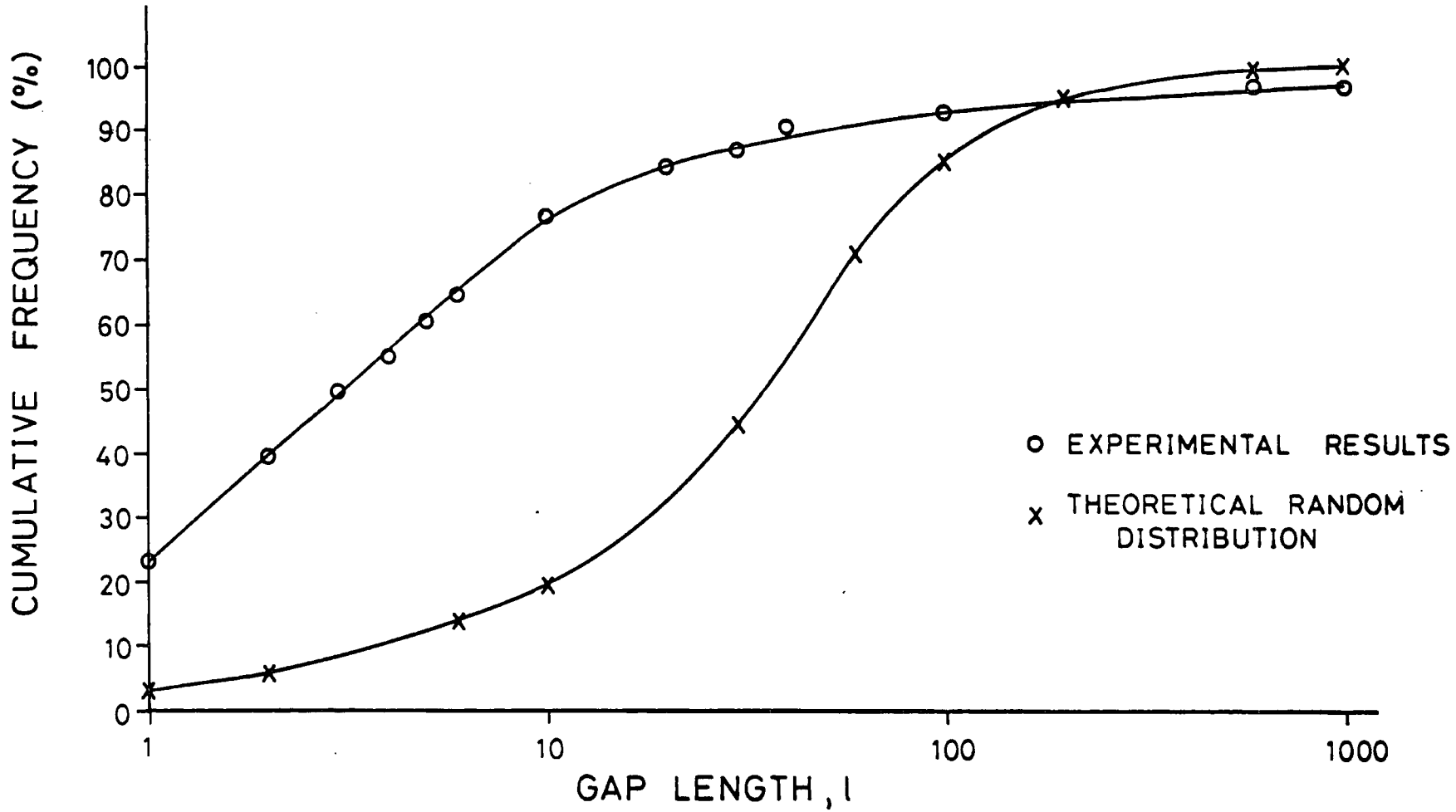


FIGURE 8.11(b). DISTRIBUTION OF ERROR-FREE INTERVALS.
(BER = 1.96×10^{-2})

It is proposed that the following constraint be appended to the above definition: All errors contained within the burst must be separated by an error-free interval of w bits or less where $w = 2/\Delta - 1$. This ensures a more even distribution of errors within the burst. In this experiment, Δ was chosen such that an error burst would cause consecutive character errors to occur in an uncoded 7-bit ASCII character stream. For characters preceded by a single start bit and terminated with a single stop bit the burst error density criterion is $2/9 = 0.22$.

The distribution of error bursts complying with the amended definition was plotted for data averaged over a 24 hour observation period and is shown in figure 8.12. It can be seen that the distribution is an exponential decay apart from a noticeable peak at b=10. The unusually high proportion of error bursts of this length was attributed to regular noise bursts from locally situated machinery which were consistently observed throughout the period. Observations on an oscilloscope indicated that these were of 0.14s in duration, ie. approximately ten times the signal element duration. The averaged measured BER over the observation period was 1.35×10^{-2}.

8.8.2 Coding performance

The performance of the HF data link in terms of bit error rate has been plotted in figure 8.9 for a 24 hour observation period. The results shown illustrate the performances obtained with no coding, with forward error correction (FEC) and with FEC and interleaving of the codewords to a depth of 16. It can be seen that the improvement with interleaving exceeds the

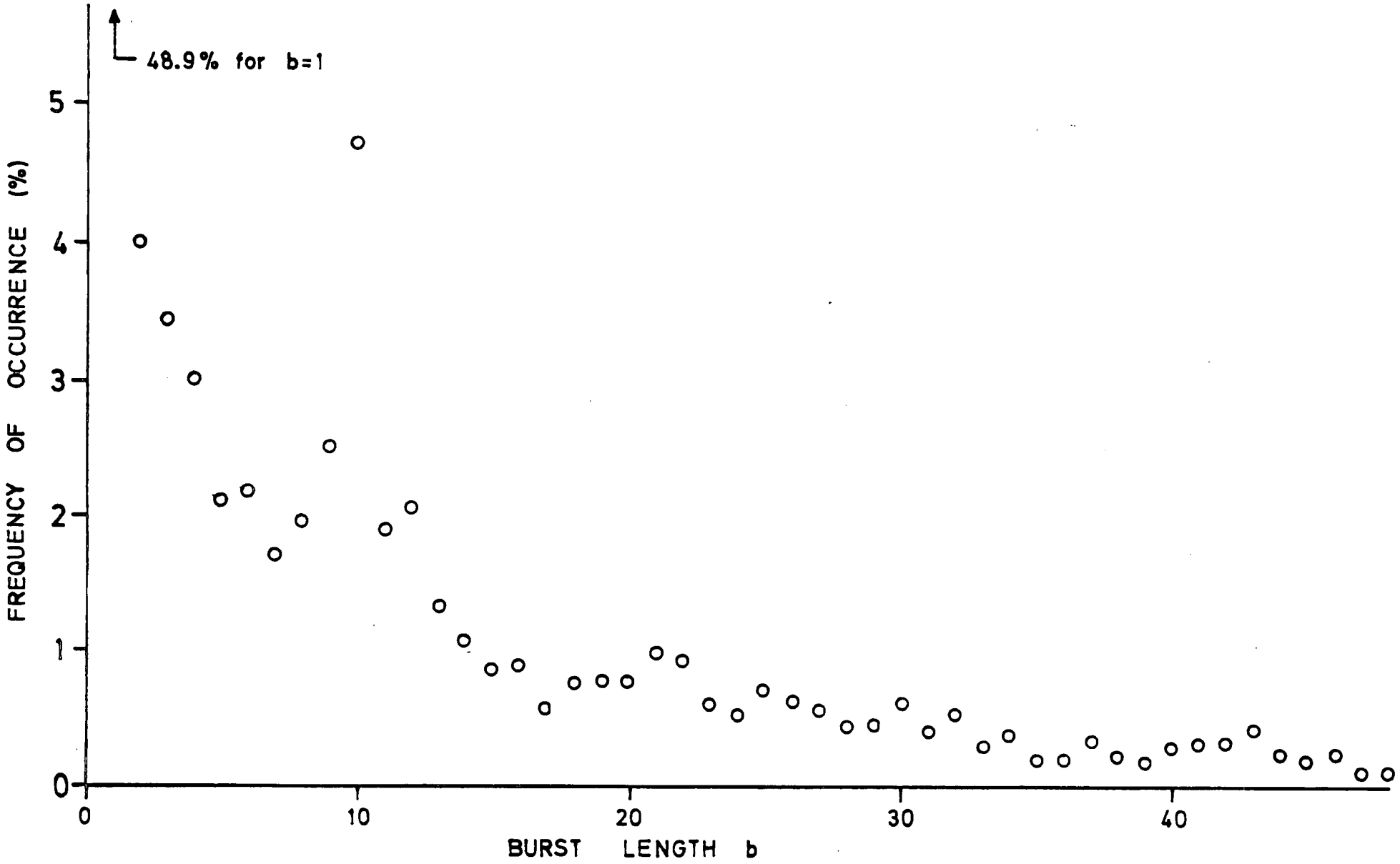


Fig.8.12. DISTRIBUTION OF ERROR BURSTS (b>1)

improvement with coding alone. Figure 8.13 shows the time distribution of the proportion of total errors corrected, with and without codeword interleaving, for a different observation period. The performance with interleaving always exceeds the performance with coding alone, but the relative improvement gained is not constant. This is also evident from fig 8.9. We shall define the "interleaving gain" as:

$$20 \log_{10}(c_i(t)/c_c(t))$$

where $c_i(t)$ and $c_c(t)$ are the percentages of total errors corrected with and without interleaving respectively. A plot of this variable against time together with a plot of BER (figure 8.14) shows that it is not necessarily dependent on error rate. The interleaving gain varies between 0.2dB and 7.2dB. It appears from these results that the nature of the error structure varies with time and, for the observation period chosen, the errors tend to cluster during the night and are of a more random nature during the day. The apparent increase in the burst nature of the error structure during the night is attributed to an increase in the skip distance which exposes the receiver to static bursts and interference from further afield. During the day, short noise spikes were observed which gave rise to an increase in the number of random errors.

Plots were made of the proportion of errors corrected against bit error rate (figure 8.15), (a) for no interleaving of codewords and, (b) for an interleaving depth of 16. The same data (over 24 hours) was used for each plot. It can again be seen that the performance with coding and interleaving exceeds the performance

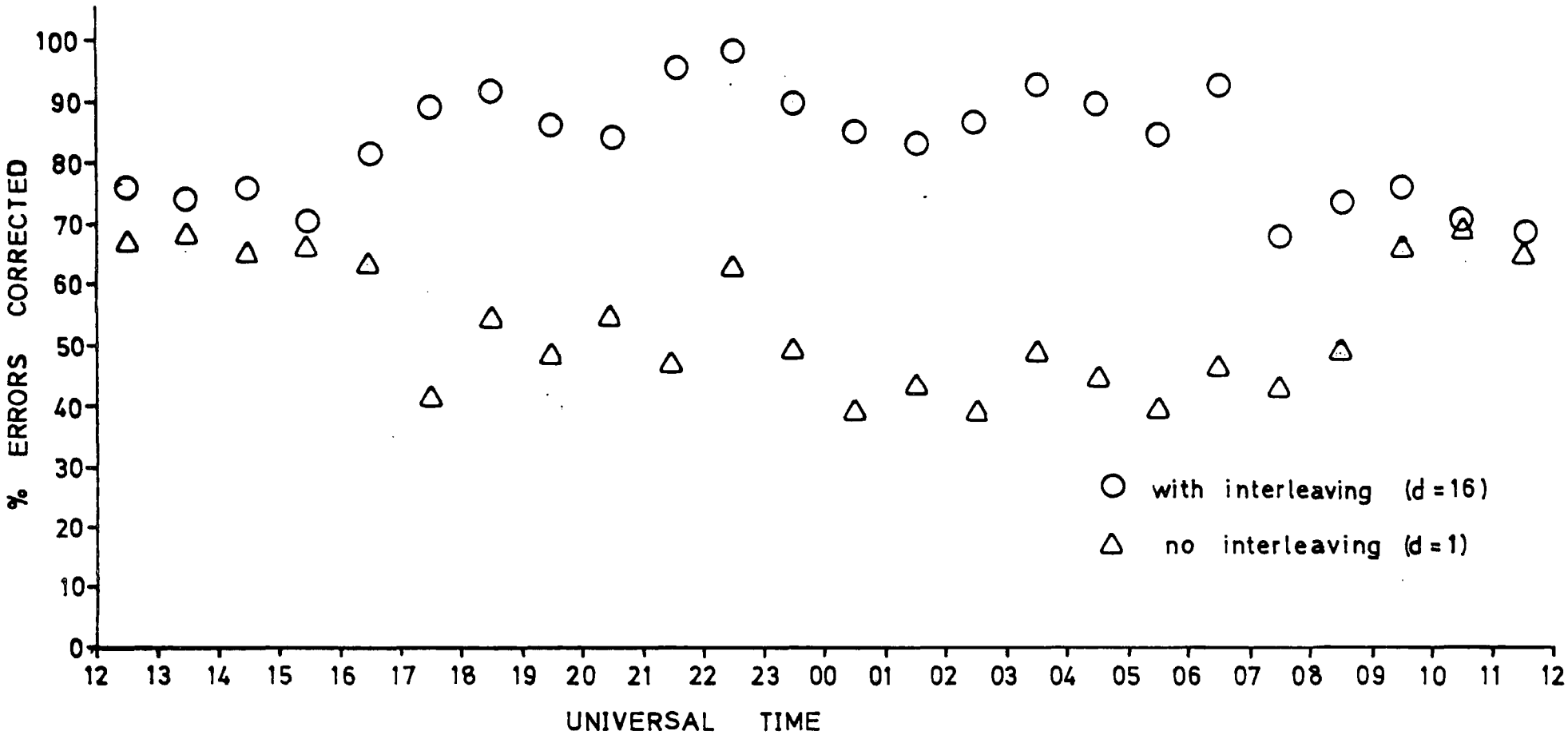


Fig. 8.13. TYPICAL DISTRIBUTION OF ERRORS CORRECTED

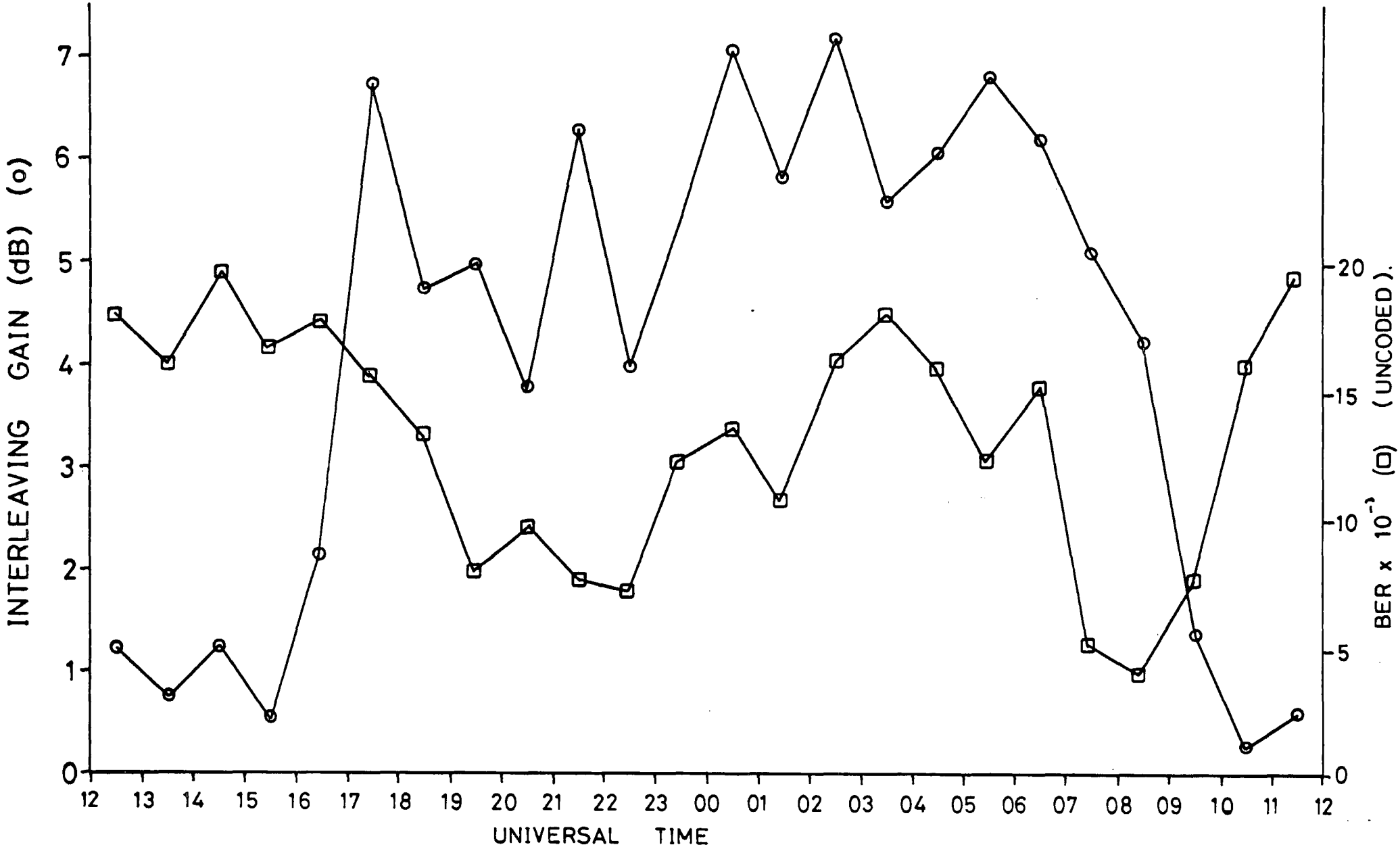
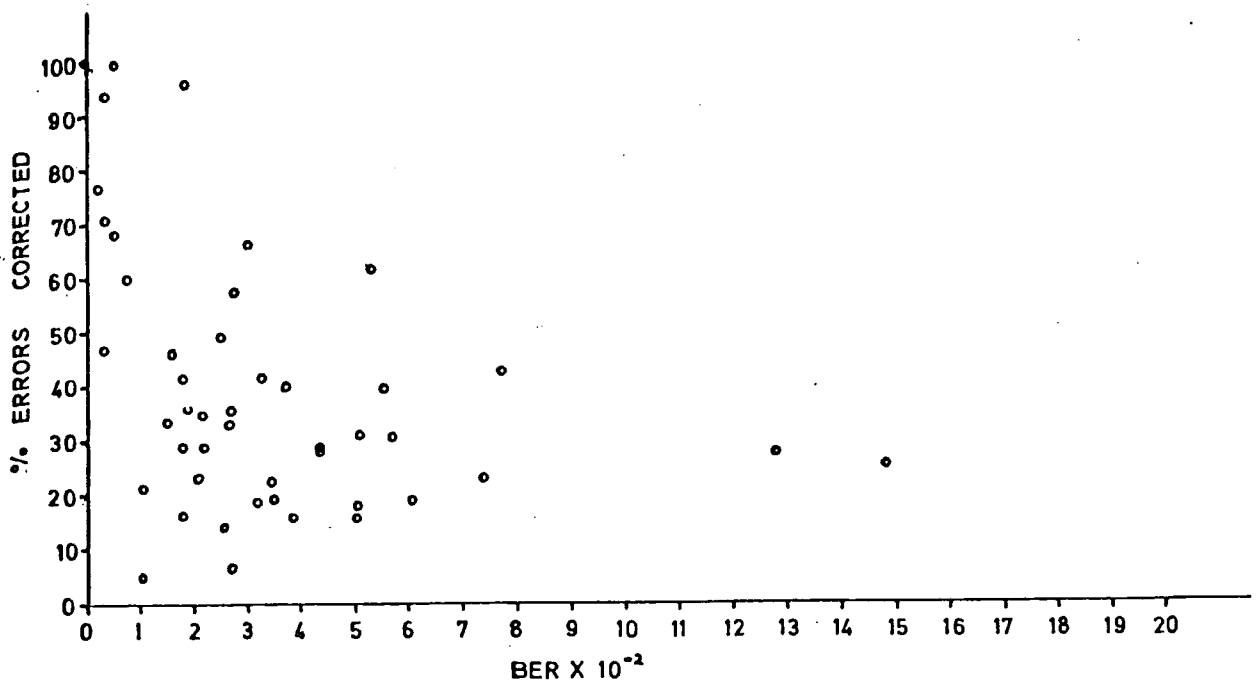
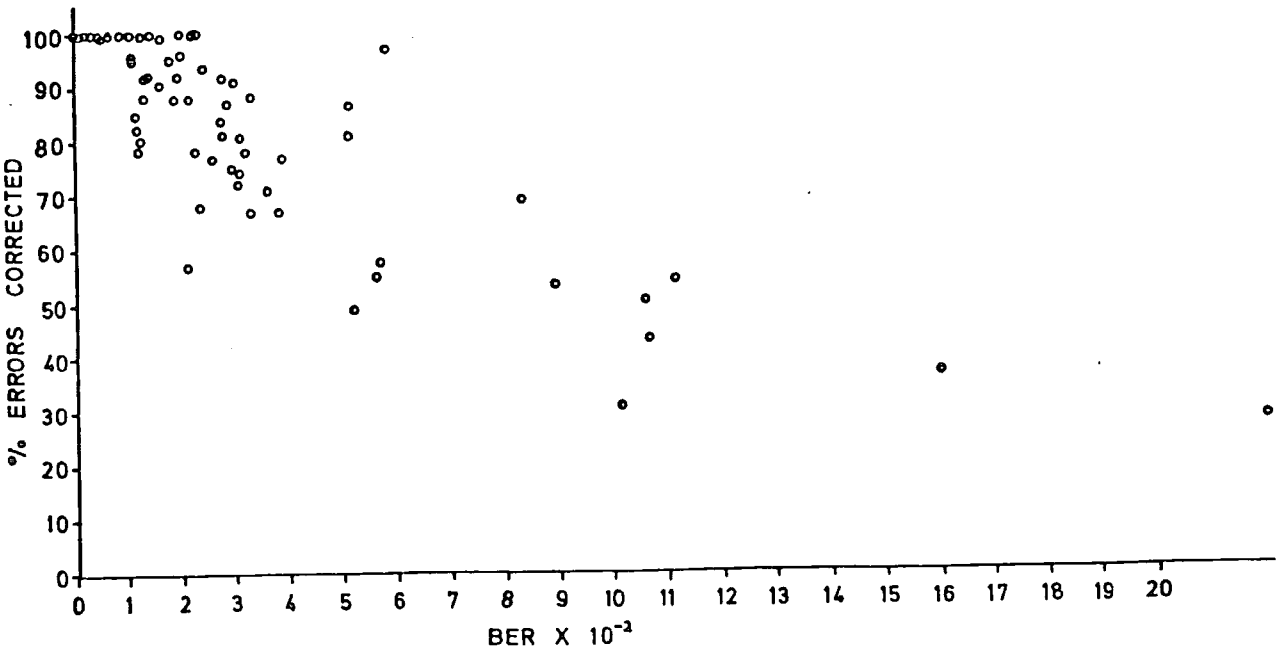


FIGURE 8.14 TIME VARIATION OF INTERLEAVING GAIN AND BER.



(a) interleaving depth = 1 (no interleaving)



(b) interleaving depth = 16

FIG. 8.15. DISTRIBUTIONS OF ERRORS CORRECTED PER MESSAGE

with coding alone, as the points tend to cluster towards the top of the plot in the latter case, and are more widely dispersed for the former.

Theoretical distributions were plotted, using the recorded data, of the proportion of errors that would be corrected for different depths of codeword interleaving. Three curves were plotted from data obtained from three intervals of 1 hour, each of which exhibited a different BER. The results were based on the assumption that if two or fewer errors occurred in a 15-bit codeword the errors would be corrected otherwise none would be corrected. Points were plotted for five values of bit interleaving. It can be seen that the relative improvement gained by using greater interleaving depths tends to decrease with increasing depth and that the curves tend to flatten for low values of BER (figure 8.16). As the interleaving depth is increased, the delay before decoding is increased. An interleaving depth of 16 appears to present a reasonable compromise between error-correcting ability, decoding delay and implementation complexity.

8.9 Conclusion

This chapter has described an experiment using microprocessor techniques undertaken to investigate the error patterns occurring over a medium-haul HF data link. The results have confirmed the view that the error statistics are significantly non-random in nature and a comparison has been made indicating precisely the deviation from the theoretical random distributions. It has also been shown that the distribution of consecutive errors and error

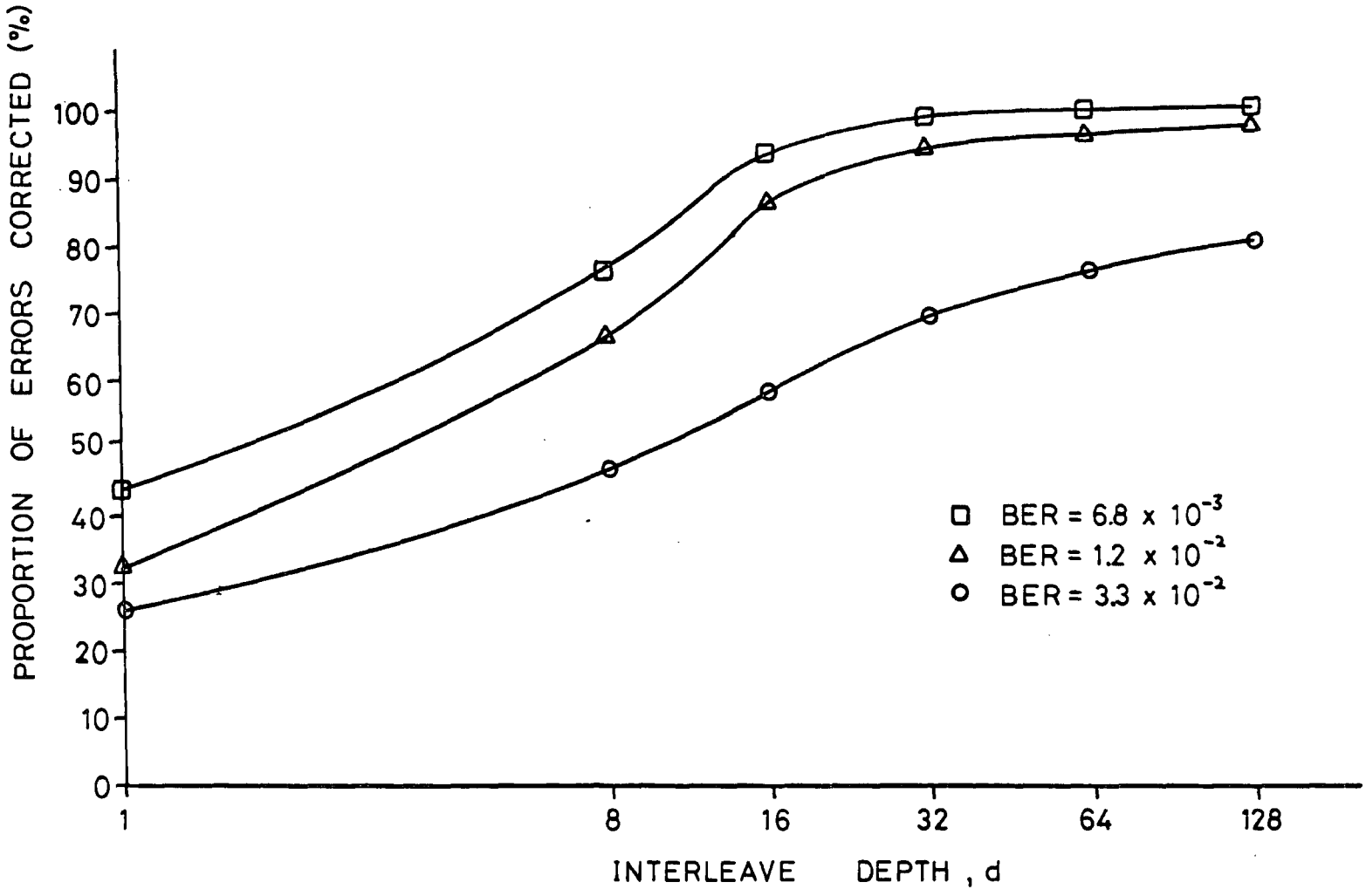


FIGURE 8.16. PROPORTION OF ERRORS CORRECTED FOR VARYING DEPTH OF CODEWORD INTERLEAVING.

free intervals appears to be independent of error rate. An investigation of error burst statistics for this experiment has shown that the distribution of burst lengths is exponential but that bursts of a fixed length may occur more frequently than expected.

An error-correcting coding scheme has been investigated using real-time encoding and decoding procedures implemented on a low-cost conventional 8-bit microprocessor. A substantial improvement in error performance was acheived by interleaving successive codewords, but the improvement in error rate was not proportional to interleaving depth. Also, the benefit gained by interleaving codewords may not be constant, indicating that the error structure is time-varying. The (15,7) BCH code interleaved to depth 16 was found to be a good compromise between error-correcting ability over the HF channel, decoding delay, and implementation complexity. In view of this, the code was chosen as the forward error correction scheme for use with the HF data modem described in chapter 7.

CHAPTER 9 Conclusion

Shortly after the advent of digital communications over wire links, investigations were carried out into the possibility of using the refractive properties of the ionosphere as a medium for data transmission over long distances. By the early 1950's, telegraphy and telephony were well established methods for communication via HF radio links, but considerable difficulties were encountered when attempts were made to use the ionospheric medium for digital data transmission.

Acceptable error rates could be achieved only if the transmission rate was severely restricted to avoid the intersymbol interference caused by long differential multipath delays. Even so, it was found that the effects of noise, often tolerated by telephony and telegraphy systems (which are highly redundant), were to introduce large numbers of errors, occasionally of such severity as to render the channel useless for many communications purposes.

Attempts were made to find techniques which could, to some extent, overcome the detrimental effects of the HF channel. Coding, diversity, time-to-frequency division mutiplexing and equalisation were all found to be effective in reducing the error rate under certain conditions, and are techniques which have been adopted in a variety of systems. Development work on systems which exhibit a certain degree of tolerance to the channel distortions was carried out in the 1950's and 60's, mainly in the U.S., and resulted in the production of a number of HF

communications terminals for military applications, notably Kineplex (5), Kathryn (6) and Codec (15). These systems were highly complex, bulky and extremely expensive to produce, and their performance was not entirely satisfactory.

Following the introduction of satellite communications during the 1960's, work in the HF field was largely abandoned in favour of this more attractive medium, which could provide reliable, high-speed data communications over long distances, albeit at high cost. However, the subsequent development of sophisticated weaponry and jamming systems led to a realisation that the satellite medium was an extremely vulnerable one, and attention was again turned to ionospheric data communications, which was often the only alternative to satellites for mobile communication over long distances. Much of the new work was concentrated on sounding techniques (11), in order to gain a better understanding of the effects of the HF channel on a digital signal. Adaptive techniques were investigated (60) to cope with the time-varying properties of the transmission medium.

The revolutionary developments in electronics technology over the last decade, notably the advances in digital electronics, have considerably eased the burden of designing complex systems for data communications (1,2). Additionally, the economic benefits to be gained using VLSI techology are often considerable. It is only in the last five years that attention has been concentrated on the possibility of implementing real-time signal processing techniques using microprocessors although such techniques have been applied mainly to line and to

VHF/UHF transmission systems (56-58). Relatively little work has been published on the direct applications of microprocessors in the field of HF data communications, which is suprising, as the inherently flexible nature of microprocessor systems make them extremely suitable for HF radio systems, where a high level of adaptability is often required.

This thesis has described an investigation into the applications of microprocessors in a variety of HF radio communications techniques. A real-time channel evaluation system has been described whereby an HF radio voice channel may be assessed as to its suitability for communication over a link. The implementation of forward error correction schemes suitable for overcoming the effects of burst errors occurring over a link have been investigated and demonstrated. This has led to the development of an adaptive modem, which uses a technique of adaptive time-to-frequency division multiplexing, together with forward error correction, to implement a low-cost system in which all the signal processing techniques have been implemented digitally. The real-time processing requirement has resulted in the development of a novel distributed processing arrangement in which a number of subsidiary ("slave") processors are under control of a central ("master") processor; the overall processing power of the system is thereby increased. Some considerable attention has been paid to the signal processing requirements of HF systems, for spectral analysis, modulation, and demodulation. This has included investigations into Fourier transformations and matched filter detection using microprocessor techniques, in addition to a study of the chirp-z transformation using a charge

coupled device.

It is hoped that this thesis has illustrated the potential of the applications of VLSI technology to the implementation of effective, low-cost systems in the field of digital HF radio communications. The complex signal processing techniques required for channel evaluation and communication over the HF medium may be realised at low cost using primarily software-based techniques. It is anticipated that further advances in the field will result in a considerable resurgence of interest in long-distance communications over HF links.

Note on publications by the author:-

"Software Cassette Interface for the SWTP 6800 Microprocessor System". Isaac, D.R. Personal Computer World, August 1979.

"Real-time Adaptive Coding of Ionospherically Propagated Data". Isaac, D.R. and Spracklen, C.T. IEEE Zurich International Seminar on Digital Communications, March 1980.

to be published:-

"A Microprocessor Implemented Trackside Recorder for British Rail". Isaac, D.R., Spracklen, C.T. & Manning, J. Journal of the Permanent Way Institution. December, 1981.

"Error Patterns and Real-time Correction Procedures for Data Transmission over HF Radio Links". Isaac, D.R. and Spracklen, C.T. IEE Conference on HF Comm. Systems and Techniques, 15-16 February, 1982.

APPENDIX 1

REFERENCES

(1) "Digital Transmission for Radio Systems". Colavito, C. Telecomm. journal. vol. 45, July 1978.

(2) "LSIs in Data Networks". van der May, J.E., Forney, G.D. & Stearns, R.W. Data Processing journal. April 1977, pp. 30-31.

(3) "Experimental Confirmation of an HF Channel Model". Watterson, C.C., Juroshek, J.R., & Bensema, W.D. IEEE Trans. Comm. Tech. Vol. COM-16, no. 6. December 1970.

(4) "Real-time Updating of MUF Predictions". Jones, T.B., Spracklen, C.T., & Stewart, C.P. AGARD Conference Proceedings no. 238, pp. 17-1 to 17-10.

(5) "Kineplex. A Bandwidth Efficient Binary Transmission System". Mosier, R.R. & Clabough, R.G. AIEE Trans. Comm. & Electron. vol. 76 pp. 723-728. 1958.

(6) "The AN/GSC-10 (KATHRYN) Variable Data Rate Modem for HF Radio". Zimmerman, M.S. & Kirch, A.L. IEEE Trans. Comm. Tech. vol. COM-15 part 2. pp. 197-204. 1967.

(7) "Equalisation of Radiolink Channels with Multipath Propagation". Castel, A., Henry, M., Rolland, P., Proc. International Seminar on on Digital Comm. IEEE cat. no. 80CH1521-4 COM. Zurich, 1980.

(8) "Telecommunications". Brown, J., & Glazier, E.V.D. Chapman & Hall, 1974.

(9) "Radio Communication Handbook". RSGB publications, 1979.

(10) "The Improvement of Slow Rate FSK by Frequency Agility and Coding". Gott, G.F. & Hillam, B. Proc. IEE Conf. of Recent Advances in HF Communications Systems & Techniques, Feb. 1979, pp. 19-24.

(11) "CW Sounding and its use for control of HF Adaptive Systems". Betts, J.A., Ellington, R.P., Jones, D.R.L. Proc IEE, vol 117 no. 12, December 1970. pp. 2209-2215.

(12) "Error Correcting Codes". Peterson, W.W. & Weldon, E.J.

(13) "Introduction to the Analysis and Processing of Signals". Lynn, P.A. Macmillan Press, 1973.

(14) "Medium-speed Digital Data Transmission over HF Channels". Darnell, M. Loughborough Conference on Digital Processing of Signals in Communications. Sept. 1977. pp. 293-402.

(15) "A Combined Coding and Modulation Approach for Communication over Dispersive Channels". Chase,D. IEEE Trans. Comm. Tech. Vol. COM-21, no. 3, March 1973.

(16) "Digital Signal Processing". Peled, Liu. Wiley Inc., 1976.

(17) "Motorola M6800 Microprocessor Programming Manual". Motorola Publications, 1978.

(18) "SWTP Computer Systems". South-West Technical Products, San Antonio, Texas.

(19) "High Speed Software Cassette Interface for the SWTP 6800 System". Isaac, D.R. Personal Computer World, August 1979. pp. 39-43.

(20) "Microcomputer Components". Motorola Publications, 1979.

(21) "Engineering Note 100". Motorola Publications, 1978.

(22) "Digital Signal Processing". Oppenheim, A.V. & Schafer, R.W. Prentice-Hall Inc. 1975.

(23) "Digital Signal Processing". Stanley. Reston, 1975.

(24) "Introduction to Digital Filters". Bogner, R.E. & Constantinides, A.G. Wiley Press, 1975.

(25) "An Algorithm for the Machine Calculation of Complex Fourier Series". Cooley, J.W. & Tukey, J.W. Math. Computation, Vol. 19, 1965, pp. 297-301.

(26) "The Chirp-Z Transform Algorithm". Rabiner, L.R., Schafer, R.W., Roder, C.M. IEEE Trans. Audio Electroacoust., Vol. AU-17, June 1969.

(27) "A Linear Filtering Approach to the Computation of the Discrete Fourier Transform". Bluestein, L.I. IEEE Trans. Audio Electroacoust., Vol. AU-18, December, 1970.

(28) "M6800 Microprocessor Applications Manual". Motorola Publications, 1975.

(29) "Comparison between the CCD Chirp-Z Transform and the Digital FFT". Buss, D.D., Broderson, R.W., Hewes, C. Proc. 1975 Naval Electronics Lab. Center Int. Conf. on Applications of CCDs.

(30) "A 500-stage CCD Transversal Filter for Spectral Analysis". Broderson, R.W., Hewes, C.R., Buss, D.D. IEEE Trans. Electronic Devices vol ED-23 pp. 143-152. Feb. 1976.

(31) "The ABCs of CCDs". Kosonocky, W.F. & Sauer, J. Electronic Devices, vol. 23 pp. 58-63, April 1975.

(32) "Real Time Spectral Analysis using Hardware Fourier and Chirp-Z Transformations". Benjamin, R. "Radio & Electronic Engineer". vol. 49, no. 2. pp. 101-107. Feb. 1979.

(33) "Communications Applications of CCD Transversal Filters". Buss, D.D., Broderson, R.W., Hewes, C.R., Tasch, A.F. IEEE Nat. Telecomm. Conf. Record, vol. 1. Dec. 1-3, 1975. pp. 1.1-1.5.

(34) "Interfacing a Hardware Multiplier to a General-Purpose Microprocessor". Davies, A.C. & Fung, Y.T.

(35) "A High-Speed FFT Processor". Zaheer, M. Ali. IEEE Trans. Comm. Vol. COM-26, no. 5. May 1978.

(36) "The Fadeogram, a sonogram-like display of the time-varying frequency response of HF SSB radio channels". Filter, J.H.J, Arazi, B., Thomson, R.G.W. IEEE Trans. Comm. June 1978 pp. 913-917.

(37) "The Fadeogram - Applied to analysing HF data modem performance". Thomson, R.G.W. & Filter, J.H.J. Proc. IEE Conf. on Recent Advances in HF Communications Systems & Techniques. Feb. 1979. pp. 19-24.

(38) "M6800 Microprocessor Applications Manual". Motorola Publications, 1975.

(39) "The M6800 Microprocessor Unit". Motorola Data Sheet.

(40) "Cyclic Codes for Errror Detection". Peterson, W.W., Brown, D.T. Proc. IRE Jan. 1961. pp. 228-235.

(41) "Principles of Data Communications". Lucky, R.W., Salz, J., Weldon, E.J. McGraw-Hill, 1968.

(42) "Further Results on Error Correcting Binary Group Codes". Bose, R.C. & Ray-Chaudhuri, D.K. IEEE Trans. Inf. & Control. Vol. IC-3, 1960. pp. 279-90.

(43) "Encoding and Error Correction Procedures for the BCH Codes". Peterson, W.W. IRE Trans. Inf. Theory. September, 1960. pp. 459-470.

(44) "Cyclic Decoding Procedures for BCH Codes". Chien, R.T. IEEE Trans. Inf. Theory. October, 1964. pp. 357-363.

(45) "On Decoding Binary BCH Codes". Berlekamp, E.R. IEEE Trans. Inf. Theory, vol. IT-11, no. 4. October, 1965.

(46) "Computer Implementation of Decoders for Several BCH Codes". Michelson, A.M. Polytech. Inst. of Brooklyn int. symposium on computer proc. in comm. April, 1969.

(47) "Practical Algebraic Decoders". Berlekamp, E.R. IEEE Trans. Inf. Theory.

(48) "Processor Implemented Decoders for Block & Convolutional Codes". Michelson, A.H. & Boyd, T.H. Proc. EASCON '74. pp. 353-360.

(49) "Generalised Burst-Trapping Codes". Burton, H.O., Sullivan, D.D., Shih Yung Tong. IEEE Trans. Inf. Theory. Vol. IT-7, no. 6, Nov. 1971.

(50) "Burst-Trapping Techniques for a Compound Channel". Shih Yung Tong. IEEE Trans. Inf. Theory. Vol. IT-15, no. 6. Nov. 1969.

(51) "Improving the Reliability of HF Data Transmissions". Sloggett, D.R. Proc. Conf. on Recent Advances in HF Comm. Systems & Techniques. Feb. 1979. pp. 74-78.

(52) "Improved Signal Design for Fading Channels by Coding". Chase, D. CNR Inc. Newton, Mass..

(53) "Characteristics of Interfering Signals in Aeronautical HF Voice Channels". Gott, G.F., Staniforth, M.J.D. Proc. IEE vol. 125, no. 11. Nov. 1978. pp. 1208-1212.

(54) "An HF Data Modem with in-band Frequency Agility". Darnell, M. IEE Proc. Conf. on Recent Advances in HF Communications Systems & Techniques. Feb. 1979. pp. 19-24.

(55) "Data at Twice the Speed Eases HF Traffic Jam". Porter, G.C., Gray, M.B., Perkett, C.E. "Electronics" journal. October 1967. pp. 115-120.

(56) "A 4800 bps Microprocessor Data Modem". Watanabe, K., Inoue, K., Sato, Y. IEEE Trans. Comm. Tech. vol. COM-26 no. 5. May 1978.

(57) "An Experimental Microprocessor-Implemented 4800 bps Limited Distance Voiceband PSK Modem". Stroh, R.W. IEEE Trans. Comm. Tech. vol. COM-26, no. 5. May 1978. pp. 507-512.

(58) "Microprocessor Implementation of Tactical Modems for Data Transmission over VHF Radio". Davies, B.H. & Davies, T.R. "Radio & Electronic Engineer", vol. 49, no. 4. pp. 204-210. April, 1979.

(59) "Microprogrammed HF Modem". Smith, S.D., Dimond, K.R., Farrell, P.G. IEE Conference on the Impact of new LSI techniques on Comm. Systems., Oct. 1980.

(60) "Data Transmission with Variable-Redundancy Error Control over an HF Channel". Goodman, V.M.F. & Farrell, P.G. Proc. IEE. Vol. 122, no. 2. February, 1975.

(61) "Real-time Adaptive Coding of Ionospherically Propagated Data." Isaac, D.R. & Spracklen, C.T. Proc. International Seminar on Digital Communications. IEEE cat. no. 80CH1521-4 COM. pp. G5.1 - G5.6. March 1980.

(62) "Selective Fading Limitations of the Kathryn Modem and some Design Considerations". Bello, P.A. IEEE Trans. Comm. Tech. vol. 13, no. 3. September 1965.

(63) "Field Test Results of the AN/GSC-10 (KATHRYN) Digital Data Terminal". Kirsch, A.L., Gray, P.R. Hanna, D.W. IEEE Trans. Comm. Tech. Vol. COM-17, no. 2. April, 1969.

(64) "Voiceband 4-Phase Data Set for Global Communications". Horlacher, R.L. & Schroeder, H.C. IEEE Trans. Cmm. Tech. Vol. COM-17 no. 3. June 1969.

(65) "Error Performance of Quadrature Pilot-tone Phase Shift Keying". Bussgang, J.J. & Leiter, M. IEEE Trans. Comm. Tech. Vol. COM-16. no. 4, August 1968.

(66) "Digital Angle Modulation". Thompson, R. & Clouting, D.R. "Wireless World", December 1976. pp. 71-76.

(67) "An Integrated HF Error-Controlled Modem". deLellis, J. & Michelson, A. Proc. 11th International Conference of Communications. San Fransisco, Calif. June 1975. pp. 13-6 to 13-10.

(68) "A Program Controlled HF Modem Implementation". deLellis, J., Fast, D., Michelson, A., Carmichael, W.. EASCON '74 Record, Washington D.C. October 1974. pp. 349-352.

(69) "Design and Performance of a new Adaptive Serial Modem on a Simulated Time-variable Multipath HF Link". IEEE Annual Conference Record, Boulder, June 1975, pp. 769-773.

(70) "Use of Pilot Tones for Real Time Channel Estimation of HF Data Circuits". Betts, J.A., Broom, R.S., Cook, S.J., Clark, J.G. Proc. IEE, vol. 122, no. 9, Sept. 1975.

(71) "Principles of Digital Data Transmission". Clark, A.P. Pentech Press, 1976.

(72) "The Improvement of Digital HF Communications through Coding". Brayer, K. IEEE Trans. Comm. Tech. Vol. COM-16., no. 6. Dec. 1968.

(73) "Application of Forward-Error Correction to a Rayleigh Fading Communication Channel". Treciokas, R. Proc. IEE, Vol. 125, no. 3. March, 1978.

(74) "Effective Application of Forward Error Control Coding to Multichannel HF Data Modems". Pierce, A.W., Barrow, B.B., Goldberg, B., Tucker, J.R. IEEE Trans. Comm. Tech. Vol. COM-18, No. 4, August, 1970.

(75) "Soft-decision Error Control for HF Data Transmission". Farrell, P.G. IEE Proc. Vol. 127, pt. F. no. 5 October 1980.

(76) "Disk Systems" manual. Smoke Signal Broadcasting Corporation, Westlake Village, Calif., USA.

(77) "Error Patterns Measured on Transequatorial HF Communications Links". Brayer, K. IEEE Trans. Comm. Tech., Vol. COM-16, no. 2. April, 1968. pp. 215-221.

APPENDIX 2

PROGRAM LISTINGS

listings for chapter 2

(1) BASIC FFT

(2) Assembler FFT

```
LIST

1 INPUT "NO. OF DATA POINTS AS POWER OF 2";S
2 N=2^S-1: M=2^S
3 DIM A(2,M):PI=3.14159
4 PRINT FRE(0); "BYTES FREE"
10 FOR I=1 TO 2^S
15 FOR J=1 TO 2
20 A(J,I)=(PEEK(K)/255-0.5
31 K=K+1
35 NEXT J:NEXT I
50 INPUT "INPUT DATA PRINTOUT";A$
55 IF A$="YES" THEN 100
60 IF A$="NO" THEN 120
70 PRINT "INVALID REPLY..YES-NO ONLY":GOTO 50
100  K=4:GOSUB 1990:PRINT"FAST FOURIER TRANSFORM"
101 PRINT"---- ------- ----------":GOSUB 1990
102 PRINT "ORIGINAL DATA" :PRINT
111 K=5:PRINT:PRINT:PRINT:PRINT:PRINT
112 PRINT "N","DATA","N","DATA"
113 PRINT "-","-----","-","-----"
114 PRINT
115 J=1:FOR I=1 TO N:PRINT J,A(1,I),J+1,A(2,I):J=J+2:NEXT I
116 GOSUB 1990
120 INPUT "APPLY WINDOW FUNCTION";A$:IF A$="YES"THEN 123
121 IF A$="NO" THEN 129
122 PRINT "INVALID REPLY":GOTO 120
123 K=2*A:ZZ=(K-1)*2:K=-(K-1)-.5:FOR I=1 TO N:FOR J=1 TO 2
124 A(J,I)=A(J,I)*(.5*(1+COS(2*PI*K/ZZ)))
125 K=K+1:NEXT J:NEXT I
127 GOTO 50
129 K=3
140 L1=N/2
145 L2=L1+1
150 A1=A(1,1)/2
160 A2=A(2,N)/2
170 A3=A(1,1)/2
180 A4=A(1,N)/2
190 FOR I=2 TO L1
200 A5=A(1,I)/2
210 A6=A(2,I)/2
220 A7=A(1,N-I+1)/2
230 '
240 A8=A(2,N-I+1)/2
250 A9=PI*(I-1)/N
260 A(1,I)=A5+A7*COS(A9)+A8*SIN(A9)
270 A(2,I)=A6-A7*SIN(A9)+A8*COS(A9)
280 A(1,N-I+1)=A5-A7*COS(A9)-A8*SIN(A9)
290 A(2,N-I+1)=-A6-A7*SIN(A9)+A8*COS(A9)
300 NEXT I
310 FOR I =1 TO L2
320 A(1,N-I+1)=A(1,N-I)
330 A(2,N-I+1)=A(2,N-I)
340 NEXT I
350 A(1,L1+1)=A1
360 A(2,L1+1)=-A2
370 A(1,1)=A3+A4
380 A(2,1)=A3-A4
390 PRINT CHR$(7)
367 INPUT "OUTPUT DATA PRINTOUT";A$:IF A$="YES" THEN 371
368 IF A$="NO" THEN 430
369 PRINT "INVALID REPLY":GOTO 367
371 PRINT "TRANSFORMED DATA":GOSUB 1990:PRINT "N","REAL","IMAG"
372 PRINT "-","-----","-----"
373 PRINT
380 N=2^S
390 FOR I=1 TO N
400 PRINT I,A(1,I),A(2,I)
410 NEXT I
420 K=11:GOSUB 1990
430 INPUT "OUTPUT DATA PLOTOUT";A$
432 IF A$="YES" THEN 435
433 IF A$="NO" THEN GOSUB 2410
434 PRINT "INVALID REPLY":GOTO 430
435 M1=-156:M2=156
436 M3=M1:M4=M2
437 FOR I=1 TO N
440 IF A(1,I)>M1 THEN M1=A(1,I)
441 IF A(1,I)<M2 THEN M2=A(1,I)
442 IF A(2,I)>M3 THEN M3=A(2,I)
443 IF A(2,I)<M4 THEN M4=A(2,I)
444 NEXT I
450 K=5:GOSUB 1990
460 GOSUB 2000
490 STOP
500 GOSUB 1000
510 GOSUB 1500
520 N=2^S
530 L1=N/2
540 L2=L1+1
550 L3=N-1
560 A1=A(2,1)
570 A(2,1)=A(1,L2)
580 A2=A(2,L2)
590 FOR I=2 TO L1
600 A3=A(1,I)
610 A4=A(2,I)
620 A5=A(1,N-I+2)
630 A6=A(2,N-I+2)
640 A(1,I)=(A5+A3)/2
650 A(2,I)=(A4-A6)/2
660 A(1,N-I+2)=(A4+A6)/2
670 A(2,N-I+2)=(A5-A3)/2
680 NEXT I
690 FOR I=L2 TO L3
700 A(1,I)=A(1,I+1)
710 A(2,I)=A(2,I+1)
720 NEXT I
730 A(1,N)=A1
740 A(2,N)=A2
750 RETURN
1000 N=2^S
1010 FOR I=1 TO N
1020 A(1,I)=A(1,I)/N
1030 A(2,I)=A(2,I)/N
1040 NEXT I
1050 RETURN
1500 N=2^S
1510 L1=N-1
1520 L2=N/2
1530 REM INPLACE SHUFFLE OF DATA BEGINS
1540 J=1
1550 FOR I=1 TO L1
1560 IF I>=J THEN 1630
1570 A1=A(1,J)
```

```
1580 A2=A(2,J)
1590 A(1,J)=A(1,I)
1600 A(2,J)=A(2,I)
1610 A(1,I)=A1
1620 A(2,I)=A2
1630 L=L2
1640 IF L>=J THEN 1680
1650 J=J-L
1660 L=L/2
1670 GOTO 1640
1680 J=J+L:NEXT I
1690 REM SHUFFLE COMPLETE
1700 REM TRANSFORM BY DECIMATION IN TIME BEGINS
1710 FOR I=1 TO P
1720 L1=2^(I-1)
1730 L2=2^(P-I)
1740 AR3=2*PI*L2/N
1750 C1=1
1760 S1=0
1770 CP=COS(AR3)
1780 SP=SIN(AR3)
1790 FOR M=1 TO L1
1800 FOR L=1 TO L2
1810 L3=(L-1)*2*L1+M
1820 A1=A(1,L3)
1830 A2=A(2,L3)
1840 B1=A(1,L3+L1)
1850 B2=A(2,L3+L1)
1860 R1=C1*B1+C2*S1
1870 R2=-C1*S1+C2*B1
1880 A(1,L3)=R1+A1
1890 A(2,L3)=R2+A2
1900 A(1,L3+L1)=A1-R1
1910 A(2,L3+L1)=A2-R2
1920 NEXT L
1930 C2=C1*CP-S1*SP
1940 S2=S1*CP+C1*SP
1950 C1=C2
1960 S1=S2
1970 NEXT M:NEXT I
1980 RETURN
1990 FOR Z2=1 TO XX:PRINT:NEXT Z2:RETURN
2000 PRINT "TRANSFORM PLOTOUT"
2010 PRINT "--------- -------"
2020 XX=3:GOSUB 1990
2030 PRINT "SCALING DATA"
2040 PRINT
2050 PRINT "REAL MAX=";M1
2060 PRINT "REAL MIN=";M2
2070 PRINT "IMAG MAX=";M3
2080 PRINT "IMAG MIN=";M4
2090 PRINT
2100 PRINT "SYMBOLS USED...REAL>>>>(*)"
2110 PRINT "                IMAG>>>>(+)"
2120 PRINT "                AXIS>>>>(.)"
2130 PRINT "REAL DATA"
2140 L1=M1-M2:L2=M3-M4
2150 FOR I=1TO N:A(1,I)=(A(1,I)-M2)*64/L1:A(2,I)=(A(2,I)-M4)*64/L2:NEXT
2160 FOR I=1 TO 64:PRINT ".";:NEXT I:PRINT
2170 M2=ABS(M2/L1):M4=ABS(M4/L2):M2=M2*64:M4=M4*64
2180 FOR I=1 TO N
2190 IF INT(M2)=INT(A(1,I)) THEN PRINT TAB(M2);"*":GOTO 2220
2200 IFINT(M2)>INT(A(1,I))THENPRINTTAB(A(1,I));"*";TAB(M2);".":GOTO2220
2210 PRINT TAB(M2);".";TAB(A(1,I));"*"
2220 NEXT I
2225 XX=5:GOSUB 1990
2230 PRINT "IMAG DATA"
2240 FOR I=1 TO 64:PRINT ".";:NEXT I:PRINT
2250 FOR I=1 TO N
2260 IF INT(M4)=INT(A(2,I)) THEN PRINT TAB(M4);"+":GOTO 2300
2270 IFINT(M4)>INT(A(2,I))THENPRINTTAB(A(2,I));"+";TAB(M4);".":GOTO2300
2280 PRINT TAB(M4);".";TAB(A(2,I));"+"
2300 NEXT I
2400 GOSUB 1990
2410 INPUT "POWER SPECTRUM PRINTOUT";A$
2420 IF A$="YES" THEN 2450
2430 IF A$="NO" THEN 2450
2440 PRINT "INVALID REPLY":GOTO 2410
2450 FOR I=1 TO N
2460 A(1,I)=A(1,I)^2+A(2,I)^2
2470 NEXT I
2475 IF A$="NO" THEN 2590
2476 GOSUB 1990
2480 PRINT "POWER SPECTRUM PRINTOUT"
2490 PRINT "------ -------- --------"
2500 PRINT "N","POWER","N","POWER"
2510 PRINT "-","------","-","------"
2520 PRINT
2530 K=1
2540 PRINT K,A(1,K),K+1,A(1,K+1)
2550 K=K+2
2560 IF K>N THEN 2580
2570 GOTO 2540
2580 PRINT
2590 INPUT"POWER SPECTRUM PLOTOUT";A$
2600 IFA$="YES" THEN 2625
2610 IF A$="NO" THEN 3000
2620 PRINT"INVALID REPLY":GOTO2590
2625 GOSUB 1990
2630 PRINT "POWER SPECTRUM PLOTOUT"
2640 PRINT "------ -------- -------"
2650 PRINT
2660 M1=-1E6:M2=1E6
2670 FOR I=1 TO N
2680 IF A(1,I)>M1 THEN M1=A(1,I)
2690 IF A(1,I)<M2 THEN M2 =A(1,I)
2700 NEXT
2710 PRINT "SCALING DATA"
2720 PRINT
2730 PRINT "MAX=";M1
2740 PRINT "MIN=";M2
2750 PRINT
2760 PRINT "SYMBOLS USED...POWER>>>>(£)"
2770 PRINT "                AXIS>>>>(.)"
2780 PRINT "POWER"
2790 FOR I=1 TO 64:PRINT".";:NEXT:PRINT
2800 L1=M1-M2
2810 FOR I=1 TO N:A(1,I)=(A(1,I)-M2)*64/L1:NEXT
2820 M2=ABS(M2/L1):M2=M2*64
2830 FOR I=1 TO N
2840 IF INT(M2)=INT(A(1,I)) THEN PRINT TAB(M2);"£":GOTO 2870
2850 IFINT(M2)>INT(A(1,I))THENPRINTTAB(A(1,I));"£";TAB(M2);".":GOTO2870
2860 PRINT TAB(M2);".";TAB(A(1,I));"£"
2870 NEXT I
2880 GOSUB 1990
3000 RETURN
```

```
                    1:         NAM   FAST FOURIER TRANSFORM
                    2:         OPT   NOG,LIS,PAG
                    3: ********************************************
                    4: *   FAST FOURIER TRANSFORM
                    5: *   ======================
                    6: *
                    7: * THIS PROGRAM COMPUTES AN N-POINT DFT OF THE COMPLEX
                    8: * SEQUENCE STORED IN THE DATA TABLE AS FOLLOWS:
                    9: *   DBASE   F(0) REAL (MSB)
                   10: *   DBASE+1 F(0) REAL (LSB)
                   11: *   DBASE+2 F(0) IMAG (MSB)
                   12: *   DBASE+3 F(0) IMAG (LSB)   ETC.
                   13: * N IS AN INTEGRAL POWER OF 2 BETWEEN 8 AND 256 INCLUSIVE.
                   14: *   R = LOG(2)N
                   15: ********************************************
                   16: *
                   17: * VARIABLE STORAGE AREA:
                   18: *
0000               19:         ORG   0
0000               20: R       RMB   1          LOG(2)N
0001               21: N2      RMB   1          NO. OF POINTS/2
0002               22: N21     RMB   1
0003               23: DINDX   RMB   2          DATA TABLE INDEX
0005               24: BINDX   RMB   2          DATA INDEX
0007               25: CINDX   RMB   2          BUTTERFLY DATA INDEX
0009               26: LKINDX  RMB   2          LOOKUP TABLE INDEX
000B               27: LKPTR   RMB   2          LOOKUP TABLE POINTER
000D               28: M       RMB   1          INDICES
000E               29: L       RMB   1
000F               30: I       RMB   1
0010               31: K       RMB   1
0011               32: J       RMB   1
0012               33: REAL    RMB   2          TEMP REAL STORE
0014               34: IMAG    RMB   2          TEMP IMAG STORE
0016               35: X       RMB   2          MULTIPLIER
0018               36: C       RMB   2          MULTPLICAND
001A               37: P       RMB   4          PRODUCT
001E               38: TB      RMB   1          TEST BYTE
                   39: *
                   40: * DATA STORAGE AREA:
                   41: *
0400               42:         ORG   $0400
0400               43: DBASE   RMB   $0400
                   44: *
                   45: * LOOKUP TABLE AREA:
                   46: *
5000               47:         ORG   $5000
5000               48: LKUP    RMB   512
                   49: *
                   50: * PROGRAM AREA:
                   51: *
0100               52:         ORG   $0100
0100 CE 0400       53:         LDX   #DBASE
0103 DF 03         54:         STX   DINDX
0105 BD 010C       55:         JSR   SHUF
0108 BD 0172       56:         JSR   FFT
010B 39            57:         RTS
                   58: ********************************************
                   59: * BIT-REVERSAL SHUFFLING ROUTINE
```



```
                   60: ********************************************
010C D6 00         61: SHUF    LDA B R
010E 5A            62:         DEC B                 R-1
010F 86 01         63:         LDA A #1
0111 48            64: FINDN2  ASL A
0112 5A            65:         DEC B                 FIND N/2
0113 26 FC         66:         BNE   FINDN2
0115 97 01         67:         STA A N2
0117 80 01         68:         SUB A #1              N/2-1
0119 48            69:         ASL A                 N-2  N-2
011A 97 02         70:         STA A N21
011C 4F            71:         CLR A
011D 97 0F         72:         STA A I               I=0
011F 97 11         73:         STA A J               J=0
0121 D6 11         74: SHUF0   LDA B J
0123 D1 0F         75:         CMP B I
0125 2F 20         76:         BLE   SHUF1
0127 BD 022E       77:         JSR   INDX4           SCALE UP (FACTOR 4)
012A DB 04         78:         ADD B DINDX+1
012C 99 03         79:         ADC A DINDX
012E D7 06         80:         STA B BINDX+1
0130 97 05         81:         STA A BINDX
0132 DE 05         82:         LDX   BINDX
0134 A6 00         83:         LDA A X               GET DATA
0136 97 12         84:         STA A REAL
0138 D6 0F         85:         LDA B I
013A BD 022E       86:         JSR   INDX4
013D DB 04         87:         ADD B DINDX+1
013F 99 03         88:         ADC A DINDX
0141 D7 08         89:         STA B CINDX+1
0143 97 07         90:         STA A CINDX
0145 DE 07         91:         LDX   CINDX
0147 A6 00         92:         LDA A X
0149 DE 05         93:         LDX   BINDX
014B A7 00         94:         STA A X
014D 96 12         95:         LDA A REAL
014F DE 07         96:         LDX   CINDX
0151 A7 00         97:         STA A X
0153 96 01         98: SHUF1   LDA A N2
0155 97 10         99:         STA A K               K=N/2
0157 96 11        100:         LDA A J
0159 91 10        101: SHUF2   CMP A K               J:=K ?
015B 2D 07        102:         BLT   SHUF3
015D 90 10        103:         SUB A K
015F 74 0010      104:         LSR   K
0162 20 F5        105:         BRA   SHUF2
0164 9B 10        106: SHUF3   ADD A K
0166 97 11        107:         STA A J
0168 96 0F        108:         LDA A I
016A 4C           109:         INC A
016B 97 0F        110:         STA A I
016D 91 02        111:         CMP A N21
016F 2F B0        112:         BLE   SHUF0
0171 39           113:         RTS
                  114: ********************************************
                  115: * FFT KERNEL:
                  116: ********************************************
0172 CE 0400      117: FFT     LDX   #DBASE          INITIALISE DATA INDEX
0175 DF 03        118:         STX   DINDX
0177 CE 5000      119:         LDX   #LKUP           AND LOOKUP INDEX
```

```
0178 DF 09      120:        STX   LKINDX
017A 4F         121:        CLR A
017B 97 0D      122:        STA A M           M=0
017D 97 0E      123:        STA A L           L=0
017F 4C         124:        INC A
0180 97 0F      125:        STA A I           I=1
                126: *
                127: * BUTTERFLY COMPUTATION:
                128: *
0182 8D 0234    129: BFLY   JSR   INDX        FIND PAIR OF POINTS
0184 D6 0D      130:        LDA B M           1ST STAGE OF COMPUTATION ?
0186 27 08      131:        BEQ   NOMULT      YES: TWIDDLE FACTOR = 1
0188 8D 0283    132:        JSR   WN          FIND TWIDDLE FACTORS
018A DE 07      133:        LDX   CINDX       BUTTERFLY ADDRESS
018C EE 00      134:        LDX   X           GET BUTTERFLY DATA (REAL)
018E DF 16      135:        STX   Y           INTO MULTIPLIER
0190 20 05      136:        BRA   ISLE1       SKIP OVER ISLAND
0192 8D 0235    137: NOMULT JSR   NOMLT
0194 20 01      138:        BRA   TSTM
0196 DE 08      139: ISLE1  LDX   LKPTR       LOOKUP ADDRESS
0198 EE 00      140:        LDX   X           GET TWIDDLE (REAL)
019A DF 18      141:        STX   Z           INTO MULTIPLICAND
019C 8D 0200    142:        JSR   MULT16      FIND C(RE)*W(RE)
019E DE 19      143:        LDX   P+1         GET PRODUCT
01A0 DF 12      144:        STX   REAL        AND SAVE IT.
01A2 DE 07      145:        LDX   CINDX       BUTTERFLY ADDRESS
01A4 EE 02      146:        LDX   2,X         GET BUTTERFLY DATA (IM)
01A6 DF 16      147:        STX   Y           INTO MULTIPLIER
01A8 8D 0200    148:        JSR   MULT16      FIND C(IM)*W(RE)
01AB DE 19      149:        LDX   P+1         GET PRODUCT
01AD DF 14      150:        STX   IMAG        AND SAVE IT
01AF DE 07      151:        LDX   CINDX       BUTTERFLY ADDRESS
01B1 EE 02      152:        LDX   2,X         GET BUTTERFLY DATA (IM)
01B3 DF 16      153:        STX   Y           INTO MULTIPLIER
01B5 DE 08      154:        LDX   LKPTR       LOOKUP ADDRESS
01B7 EE 02      155:        LDX   2,X         GET TWIDDLE (IM)
01B9 DF 18      156:        STX   Z           INTO MULTIPLICAND
01BB 8D 0200    157:        JSR   MULT16      FIND C(IM)*WN(IM)
01BE 96 19      158:        LDA A REAL        C(RE)*W(RE)
01C0 D6 13      159:        LDA B REAL+1
01C2 F0 19 02   160:        SUB B P+2         C(RE)*W(RE)-C(IM)*W(IM)
01C5 82 18      161:        SBC A P+1
01C7 47         162:        ASR A
01C8 56         163:        ROR B             DIVIDE BY 2
01C9 97 12      164:        STA A REAL
01CB D7 13      165:        STA B REAL+1
01CD DE 07      166:        LDX   CINDX
01CF EE 00      167:        LDX   X           GET BUTTERFLY DATA (RE)
01D1 DF 16      168:        STX   Y           INTO MULTIPLIER
01D3 8D 0200    169:        JSR   MULT16      FIND C(RE)*W(IM)
01D6 96 14      170:        LDA A IMAG        C(IM)*W(RE)
01D8 D6 15      171:        LDA B IMAG+1
01DA DB 12      172:        ADD B P+2         C(IM)*W(RE)+C(RE)*W(IM)
01DC 99 18      173:        ADC A P+1
01DE 47         174:        ASR A
01DF 56         175:        ROR B             DIVIDE BY 2
01E0 97 14      176:        STA A IMAG
01E2 D7 15      177:        STA B IMAG+1
01E4 8D 0268    178:        JSR   NWDATA      FIND NEW DATA
                179: *
```



```
                180: * IS M=2^(I-1)-1 ?
                181: *
01E6 8D 33      182: TSTM   BSR   INDX5       FIND 2^(I-1)
01E8 C0 01      183:        SUB B #1          -1
01EA D1 0D      184:        CMP B M           M=2^(I-1)-1 ?
01EC 27 06      185:        BEQ   TSTL        YES: TEST L
01EE 7C 000D    186:        INC   M           NO: M=M+1
01F1 7E 0184    187:        JMP   BFLY        DO NEXT BUTTERFLY
                188: *
                189: * IS L=2^(R-I)-1 ?
                190: *
01F4 7F 000D    191: TSTL   CLR   M           M=0
01F7 C6 01      192:        LDA B #1
01F9 96 00      193:        LDA A R
01FB 90 0F      194:        SUB A I           R-I
01FD 8D 22      195:        BSR   INDX6       FIND 2^(R-I)
01FF C0 01      196:        SUB B #1          -1
0201 D1 0E      197:        CMP B L           L=2^(R-I)-1 ?
0203 27 06      198:        BEQ   TSTI        YES: TEST I
0205 7C 000E    199:        INC   L           NO: L=L+1
0208 7E 0184    200:        JMP   BFLY        DO NEXT BUTTERFLY
                201: *
                202: * IS I=R ?
                203: *
020B 7F 000E    204: TSTI   CLR   L           L=0
020E 96 0F      205:        LDA A I
0210 91 00      206:        CMP A R           R=I ?
0212 27 06      207:        BEQ   ENFFT       YES: FFT COMPLETE.
0214 7C 000F    208:        INC   I           NO: I=I+1
0217 7E 0184    209:        JMP   BFLY        DO NEXT BUTTERFLY
                210: *
021A 39         211: ENFFT  RTS
                212: *
                213: * FIND 2^(I-1)
                214: *
021B C6 01      215: INDX5  LDA B #1
021D 96 0F      216:        LDA A I
021F 80 01      217:        SUB A #1          I-1
                218: *
                219: * FIND 2^(ACCA)
                220: *
0221 27 04      221: INDX6  BEQ   INDX3
0223 58         222: INDX2  ASL B
0224 4A         223:        DEC A
0225 26 FC      224:        BNE   INDX2
0227 39         225: INDX3  RTS
                226: *
                227: * SCALE UP BY FACTOR 4
                228: *
0228 4F         229: INDX4  CLR A
0229 59         230:        ROL B
022A 49         231:        ROL A
022B 59         232:        ROL B
022C 49         233:        ROL A
022D 39         234:        RTS
                235: *
                236: * LOCATE PAIR OF POINTS FOR BUTTERFLY
                237: *
022E 96 0F      238: INDX   LDA A I
0230 D6 0E      239:        LDA B L
```

```
0038 58        240: INDX1  ASL B               FIND L*2^I
0039 49        241:       DEC A
003A 26 FC     242:       BNE   INDX1
003C DB 00     243:       ADD B M              M+L*2^I
003E 8D EE     244:       BSR   INDX4          SCALE UP BY FACTOR 4
0040 DB 04     245:       ADD B DINDX+1        ADD TO BASE OF DATA TABLE
0042 99 03     246:       ADC A DINDX
0044 D7 06     247:       STA B BINDX+1
0046 97 05     248:       STA A BINDX
0048 8D 07     249:       BSR   INDX5          FIND 2^(I-1)
004A 8D E2     250:       BSR   INDX4          SCALE UP BY FACTOR 4
004C DB 18     251:       ADD B BINDX+1
004E 99 05     252:       ADC A BINDX
0050 D7 08     253:       STA B CINDX+1        BUTTERFLY LOCATION
0052 97 07     254:       STA A CINDX
0054 39        255:       RTS
               256: *
               257: * FIRST STAGE: TWIDDLE FACTOR= 1
               258: *
0055 DE 07     259: MOMLT  LDX   CINDX
0057 96 00     260:       LDA A X
0059 E6 01     261:       LDA B 1,X
005B 42        262:       MUL A
005C 56        263:       MUL B
005D D7 12     264:       STA B REAL
005F 97 13     265:       STA A REAL+1
0061 96 02     266:       LDA A 2,X
0063 E6 03     267:       LDA B 3,X
0065 42        268:       MUL A
0066 56        269:       MUL B
0067 97 14     270:       STA A IMAG
0069 D7 15     271:       STA B IMAG+1
               272: *
               273: * FIND NEW POINTS FROM OLD
               274: *
006B DE 05     275: NWDATA LDX   BINDX
006D A6 00     276:       LDA A X
006F E6 01     277:       LDA B 1,X            GET REAL DATA
0071 42        278:       MUL A
0072 56        279:       MUL B
0073 97 16     280:       STA A Y              SAVE IN Y
0075 D7 17     281:       STA B Y+1
0077 DB 13     282:       ADD B REAL+1
0079 99 12     283:       ADC A REAL           ADD TO REAL PRODUCT
007B A7 02     284:       STA A X              NEW REAL DATA
007D A6 02     285:       LDA A 2,X            GET IMAG DATA
007F E6 03     286:       LDA B 3,X
0081 42        287:       MUL A
0082 56        288:       MUL B
0083 97 18     289:       STA A Z              SAVE IN Z
0085 D7 19     290:       STA B Z+1
0087 DB 15     291:       ADD B IMAG+1         ADD TO IMAG PRODUCT
0089 99 14     292:       ADC A IMAG
008B DE 05     293:       LDX   BINDX          DATA ADDRESS
008D A7 02     294:       STA A 2,X            NEW IMAG DATA
008F E7 03     295:       STA B 3,X
0091 A6 00     296:       LDA A Y              GET OLD REAL DATA
0093 E6 17     297:       LDA B Y+1
0095 D3 13     298:       SUB B REAL+1         SUBTRACT REAL PRODUCT
```



```
0097 92 12     299:       SBC A REAL
0099 DE 07     300:       LDX   CINDX          BUTTERFLY ADDRESS
009B A7 00     301:       STA A X              NEW BFLY DATA (REAL)
009D E7 01     302:       STA B 1,X
009F 96 18     303:       LDA A Z              GET OLD IMAG DATA
00A1 D6 19     304:       LDA B Z+1
00A3 D0 15     305:       SUB B IMAG+1         SUBTRACT IMAG PRODUCT
00A5 92 14     306:       SBC A IMAG
00A7 A7 02     307:       STA A 2,X            NEW BFLY DATA (IMAG)
00A9 E7 03     308:       STA B 3,X
00AB 39        309:       RTS
               310: *
               311: * SCALE PRODUCT
               312: *
00AC 79 001C   313: SCALE  ROL   P+2
00AF 79 001B   314:       ROL   P+1
00B2 79 001A   315:       ROL   P
00B5 39        316:       RTS
               317: *
               318: * FIND TWIDDLE FACTOR
               319: * FIND M*2^(P-I)*2^(8-P) = M*2^(8-I)
               320: *
00B6 86 08     321: WN     LDA A #8
00B8 90 0F     322:       SUB A I              8-I
00BA 27 04     323:       BEQ   WN2
00BC 58        324: WN1    ASL B                FIND M*2^(8-I)
00BD 4A        325:       DEC A
00BE 26 FC     326:       BNE   WN1
00C0 D7 1E     327: WN2    STA B TB
00C2 8D 022E   328:       JSR   INDX4          SCALE BY FACTOR 4
00C5 DB 09     329:       ADD B LKINDX+1
00C7 99 08     330:       ADC A LKINDX
00C9 97 0B     331:       STA A LKPTR
00CB D7 0C     332:       STA B LKPTR+1
00CD 39        333:       RTS
               334: ****************************************
               335: * 16 BY 16 MULTIPLICATION ROUTINE USING BOOTH'S
               336: * ALGORITHM.
               337: *  MULTIPLIER: Y,Y+1
               338: *  MULTIPLICAND: Z,Z+1
               339: *  PRODUCT: P,P+1,P+2,P+3,P+4
               340: *   MULTIPLIER DESTROYED
               341: *   MULTIPLICAND PRESERVED
               342: ************************************
00CE CE 0000   343: MULT16 LDX   #0          CLEAR REGS.
00D1 DF 1A     344:       STX   P
00D3 DF 1C     345:       STX   P+2
00D5 7F 0015   346:       CLR   TB            CLEAR TEST BYTE
00D8 CE 0010   347:       LDX   #16
00DB 96 17     348: LOOP   LDA A Y+1
00DD 84 01     349:       AND A #1
00DF 16        350:       TAB
00E0 98 1E     351:       EOR A TB            Y(LSB)-Y(LSB-1) ?
00E2 27 15     352:       BEQ   SHIFT         YES: SHIFT
00E4 5D        353:       TST B
00E5 27 06     354:       BEQ   ADD           Y(LSB)=0
00E7 96 1A     355: SUBTR  LDA A P
00E9 D6 1B     356:       LDA B P+1
00EB D0 19     357:       SUB B Z+1
00ED 96 1A     358: ADD    LDA A P
```

```
0301 D6 1B     360:        LDA B P+1
0303 DB 19     361:        ADD B Z+1
0305 99 18     362:        ADC A Z
0307 97 14     363:        STA A P
0309 D7 15     364:        STA B P+1
030B 7F 001E   365: SHIFT  CLR   TB          CLEAR TEST BYTE
030E 76 0016   366:        ROR   Y
0311 76 0017   367:        ROR   Y+1
0314 79 001E   368:        ROL   TB
0317 77 0014   369:        ASR   P
031A 76 0013   370:        ROR   P+1
031D 76 0012   371:        ROR   P+2
0320 76 0011   372:        ROR   P+3
0323 09        373:        DEX                 DEC SHIFT COUNT
0324 26 E7     374:        BNE   LOOP
0326 39        375:        RTS
               376:        END
      NO ERROR(S) DETECTED
```

001:

listings for chapter 3

(1) The HF Spectrogram

```
                    1:         NAM   HF SPECTROGRAM
                    2:         OPT   NOS,LIS,PAG
                    3: ********************************************
                    4: *
                    5: *        HF SPECTROGRAM SOFTWARE
                    6: *        =======================
                    7: *
                    8: * THIS PROGRAM IS USED TO EVALUATE AND DISPLAY THE
                    9: * TIME-VARYING SPECTRAL PROPERTIES OF AN HF VOICE
                   10: * CHANNEL.
                   11: *
                   12: *      X AXIS:  TIME DOMAIN,   Q POINTS (Q <= 256).
                   13: *      Y AXIS:  FREQ DOMAIN, N/2 POINTS (N <= 256).
                   14: *
                   15: *  DATA IS SAVED ON DISC FOR SUBSEQUENT ANALYSIS
                   16: *  IF REQUIRED.
                   17: *
                   18: ********************************************
                   19: *
                   20: *  I/O ADDRESSES:
                   21: *
8010               22: XAXIS  EQU   $8010        TIME AXIS PORT
8012               23: YAXIS  EQU   $8012        FREQ AXIS PORT
8013               24: ZAXIS  EQU   $8013        POWER SPECTRUM O/P.
801A               25: INDATA EQU   $801A        INPUT DATA
8008               26: CLOCK  EQU   $8008        REAL-TIME CLOCK
                   27: *
                   28: *  MONITOR ROUTINES:
                   29: *
E04A               30: INHEX  EQU   $E04A        IN 1 HEX CHAR
E073               31: INCH   EQU   $E073
E0BF               32: OUT2H  EQU   $E0BF        OUT 2 HEX CHARS
E1D1               33: OUTCH  EQU   $E1D1        OUT 1 ASCII CHAR
E0E3               34: CNTRL  EQU   $E0E3        RESTART ADDRESS
A000               35:        ORG   $A000
A000 05 CC         36:        FDB   CLK         INTERRUPT VECTOR
                   37: *
                   38: * DISC OPERATING SYSTEM ENTRY POINTS:
                   39: *
7786               40: DFM    EQU   $7786        DISC FILE MANAGEMENT
72A9               41: ZTYPDE EQU   $72A9        TYPE DISC ERROR
7733               42: CDFM   EQU   $7733        CLOSE ACTIVE FILES
7283               43: ZWARMS EQU   $7283        WARM START
                   44: *
                   45: * MAIN PROCEDURE VARIABLE STORAGE:
                   46: *
0000               47:        ORG   0
0000               48: N      RMB   2           NO. OF POINTS
0002               49: N2     RMB   1           N/2
0003               50: P      RMB   1           LOG(2)N
0004               51: D      RMB   2           NO. OF LINES
0006               52: I      RMB   1
0007 09            53: PERIOD FCB   9           SAMPLE PERIOD CONST
0008               54: DLY    RMB   1           DELAY BETWEEN LINES
0009               55: HOURS  RMB   1           CLOCK VARIABLES
000A               56: MINS   RMB   1
000B               57: SECS   RMB   1
000C               58: XTEMP  RMB   2           XREG TEMP STORE
000E               59: XTEMP1 RMB   2
```

```
0010               60: OSD    RMB   1           DFM FUNCTION CODE
0011               61: XINC   RMB   1           X INCREMENT
0012               62: YINC   RMB   1           Y INCREMENT
                   63: *
                   64: * FFT ROUTINE VARIABLE STORAGE:
                   65: *
0013               66: M      RMB   1           BUTTERFLY INDICES
0014               67: L      RMB   1
0015               68: I      RMB   1
0016               69: K      RMB   1
0017               70: NH     RMB   1
0018 00 26         71: CLKUP  FDB   CBASE       COSINE LOOKUP POINTER
001A 01 26         72: SLKUP  FDB   SBASE       SINE LOOKUP POINTER
001C 02 26         73: WLKUP  FDB   WBASE       WINDOW LOOKUP POINTER
001E               74: BINDX  RMB   2           DATA POINTER
0020               75: CINDX  RMB   2           LOOKUP POINTER
0022               76: REAL   RMB   2           REAL COEFFICIENT
0024               77: IMAG   RMB   2           IMAG COEFFICIENT
0026               78: CBASE  RMB   256
0126               79: SBASE  RMB   256
0326               80: WBASE  RMB   512
0526               81: DBASE  RMB   512
                   82: *
                   83: * MULTIPLICATION ROUTINE VARIABLES:
                   84: *
0626               85: Y      RMB   2           MULTIPLIER
0628               86: Z      RMB   2           MULTIPLICAND
062A               87: P      RMB   4           PRODUCT
062E               88: TB     RMB   1           TEST BYTE
062F               89: TEMP   RMB   1
                   90: *
                   91: * DIVISION & SQR ROOT VARIABLES:
                   92: *
0630               93: XKDVSR RMB   2           DIVISOR
0632               94: XKDVND RMB   4           DIVIDEND
0636               95: XKQUOT RMB   4           QUOTIENT
063A               96: FLAG   RMB   1
063B               97: G      RMB   4
                   98: *
                   99: * DISC FILE CONTROL BLOCK
                  100: *
063F              101: FCB    RMB   2           FUNCTN CODE/ ERR STATUS
0641 01           102:        FCB   1           DRIVE UNIT NUMBER
0642 53           103:        FCC   'SAMPLS000' FILE NAME
064B              104:        RMB   154         REMAINDER OF FCBLOCK
                  105: *
                  106: ********************************************
                  107: *   MAIN PROCEDURE BEGINS
                  108: ********************************************
0100              109:        ORG   $0100
0100 BD 0120      110:        JSR   PIASET      INITIALISE PIA'S
0103 BD 013A      111:        JSR   PARAM       PRINT MESSAGES ON TERMINAL ETC.
0106 BD 0258      112:        JSR   INCS        CALCULATE X & Y INCREMENTS
0109 BD 031A      113:        JSR   OPNFIL      OPEN FILE FOR WRITE
010C BD 037B      114: SLOOP  JSR   ORIGIN      RETURN BEAM TO ORIGIN
010F BD 0271      115:        JSR   PLOT        PLOT A FRAME
0112 BD 02D7      116:        JSR   PTIME       PRINT TIME
0115 BD 0301      117:        JSR   PREQ        ANOTHER PLOT ?
0118 26 F2        118:        BNE   SLOOP       YES! LOOP BACK
011A BD 0381      119:        JSR   CLOSFL      CLOSE FILE
```

```
011D 7E 7333  120:         JMP   ZWARMS       JUMP TO DOS
              121: ********************************************
              122: * INITIALISE I/O PORTS:
              123: ********************************************
0120 CE FF04  124: PIASET LDX   #$FF04
0123 FF 8010  125:        STX   XAXIS
0126 FF 8012  126:        STX   YAXIS
0129 FF 8014  127:        STX   ZAXIS
012C CE 0004  128:        LDX   #$0004
012F FF 8018  129:        STX   INDATA
0132 CE 0005  130:        LDX   #$0005
0135 FF 8008  131:        STX   CLOCK
0138 0E       132:        CLI
0139 39       133:        RTS
              134: ********************************************
              135: * PRINT MESSAGES AND GET PARAMETERS FROM TERMINAL
              136: ********************************************
013A CE 013E  137: PARAM  LDX   #BANNR       PRINT BANNER
013D BD 023D  138:        JSR   PRINT
0140 CE 0159  139:        LDX   #TREQ        REQUEST FOR TIME
0143 BD 023D  140:        JSR   PRINT
0146 CE 01D9  141:        LDX   #HMES        REQUEST HOURS
0149 BD 023D  142:        JSR   PRINT
014C BD 030D  143:        JSR   BYTE         GET HOURS
014F 97 09    144:        STA A HOURS
0151 CE 01E3  145:        LDX   #MMES        REQUEST MINS
0154 BD 023D  146:        JSR   PRINT
0157 BD 030D  147:        JSR   BYTE         GET MINS
015A 97 04    148:        STA A MINS
015C CE 01EC  149:        LDX   #SMES        REQUEST SECS
015F BD 023D  150:        JSR   PRINT
0162 BD 030D  151:        JSR   BYTE         GET SECS
0165 97 0E    152:        STA A SECS
0167 CE 01F5  153:        LDX   #RREQ        GET LOG(2)N
016A BD 023D  154:        JSR   PRINT
016D BD E04A  155:        JSR   INHEX
0170 97 03    156:        STA A R
0172 BD 024A  157:        JSR   EXP          FIND N
0175 97 00    158:        STA A N
0177 D7 01    159:        STA B N+1
0179 44       160:        LSR A
017A 56       161:        ROR B
017B D7 02    162:        STA B N2           N/2
017D B6 02    163:        LDA A #SREQ        GET LOG(2)Q
017F BD 023D  164:        JSR   PRINT
0182 BD E04A  165:        JSR   INHEX
0185 97 06    166:        STA A S
0187 BD 024A  167:        JSR   EXP
018A 97 04    168:        STA A Q
018C D7 05    169:        STA B Q+1
018E CE 020F  170:        LDX   #IREQ        GET DELAY
0191 BD 023D  171:        JSR   PRINT
0194 BD 030D  172:        JSR   BYTE
0197 97 08    173:        STA A DLY
0199 BD 030D  174:        JSR   BYTE
019C 97 09    175:        STA A DLY+1
              176: ********************************************
              177: * MESSAGES:
              178: ********************************************
019E 0D       179: BANNR  FCB   $D,$A
```



```
01A0 2A       180:        FCC   '*** HF SPECTROGRAM ***'
01B8 04       181:        FCB   4
01B9 0D       182: TREQ   FCB   $D,$A
01BB 45       183:        FCC   'ENTER TIME OF DAY AS FOLLOWS:'
01D8 04       184:        FCB   4
01D9 0D       185: HMES   FCB   $D,$A
01DB 48       186:        FCC   'HOURS ?'
01E2 04       187:        FCB   4
01E3 0D       188: MMES   FCB   $D,$A
01E5 4D       189:        FCC   'MINS ?'
01EB 04       190:        FCB   4
01EC 0D       191: SMES   FCB   $D,$A
01EE 53       192:        FCC   'SECS ?'
01F4 04       193:        FCB   4
01F5 0D       194: RREQ   FCB   $D,$A,$A
01F8 4C       195:        FCC   'LOG(2)N ?'
0201 04       196:        FCB   4
0202 0D       197: SREQ   FCB   $D,$A,$A
0205 4C       198:        FCC   'LOG(2)Q ?'
020E 04       199:        FCB   4
020F 0D       200: IREQ   FCB   $D,$A,$A
0212 44       201:        FCC   'DELAY UNITS ?'
021F 04       202:        FCB   4
0220 0D       203: PMES   FCB   $D,$A
0222 41       204:        FCC   'ANOTHER PLOT ? '
0231 04       205:        FCB   4
0232 0D       206: TMES   FCB   $D,$A,$A
0235 54       207:        FCC   'TIME = '
023C 04       208:        FCB   4
              209: ********************************************
              210: * PRINT A MESSAGE:
              211: ********************************************
023D A6 00    213: PRINT  LDA A X
023F 81 04    213:        CMP A #4           DELIMITER ?
0241 27 06    214:        BEQ   PRIN1
0243 BD E1D1  215:        JSR   OUTCH        PRINT IT
0246 08       216:        INX
0247 20 F4    217:        BRA   PRINT
0249 39       218: PRIN1  RTS
              219: ********************************************
              220: * FIND 2^(ACCA)
              221: ********************************************
024A B7 062F  222: EXP    STA A TEMP
024D C6 01    223:        LDA B #1
024F 4F       224:        CLR A
0250 58       225: EXP1   ASL B
0251 49       226:        ROL A
0252 7A 062F  227:        DEC   TEMP
0255 26 F9    228:        BNE   EXP1
0257 39       229:        RTS
              230: ********************************************
              231: * FIND X AND Y INCREMENTS:
              232: ********************************************
0258 C6 01    233: INCS   LDA B #1
025A 96 05    234:        LDA A Q+1
025C 27 04    235:        BEQ   INC2
025E 58       236: INC1   ASL B
025F 49       237:        ASL A
0260 26 FC    238:        BNE   INC1
0262 D7 12    239: INC2   STA B YINC
```

```
0264 C6 01    240:        LDA B #1
0266 96 01    241:        LDA A N+1
0268 27 04    242:        BEQ   INC4
026A 58       243: INC3   ASL B
026B 48       244:        ASL A
026C 26 FC    245:        BNE   INC3
026E D7 11    246: INC4   STA B XINC
0270 39       247:        RTS
              248: ********************************************
              249: * PLOT A FRAME
              250: ********************************************
0271 DE 04    251: PLOT   LDX   0           SAVE NO. OF LINES
0273 DF 0E    252:        STX   XTEMP1      IN TEMP STORE
0275 86 02    253:        LDA A #2
0277 97 10    254:        STA A 0$0
0279 86 FF    255:        LDA A #$FF        'NEW PLOT' TO DISC
027B BD 033E  256:        JSR   PLCTRL
027E 96 09    257:        LDA A HOURS
0280 BD 033E  258:        JSR   PLCTRL
0283 96 0A    259:        LDA A MINS
0285 BD 033E  260:        JSR   PLCTRL
0288 96 0B    261:        LDA A S
028A BD 033E  262:        JSR   PLCTRL
028D 96 03    263:        LDA A P
028F BD 033E  264:        JSR   PLCTRL
0292 CE 0426  265: PLOT1  LDX   #DBASE
0295 BD 0346  266:        JSR   SAMPLE      GET DATA FROM A/D
0298 BD 0394  267:        JSR   WINDOW      MULTIPLY BY TIME WINDOW
029B BD 03F6  268:        JSR   SHUF        BIT-REVERSAL SHUFFLE
029E BD 0648  269:        JSR   FFT         FAST FOURIER TRANSFORM
02A1 BD 05F7  270:        JSR   PSPEC       FIND POWER SPECTRUM
02A4 D6 02    271:        LDA B N2          GET N/2
02A6 CE 0427  272:        LDX   #DBASE+1    BOTTOM OF TRANSFORMED DATA
02A9 37       273: DPLOT  PSH B             SAVE POINTS COUNT
02AA A6 00    274:        LDA A X           GET DATA BYTE
02AC 43       275:        COM A
02AD B7 3013  276:        STA A ZAXIS       OUTPUT TO SCOPE
02B0 BD 02FA  277:        JSR   DELAY       WAIT TO SETTLE
02B3 96 12    278:        LDA A YINC
02B5 BB 3012  279:        ADD A YAXIS       INCREMENT Y AXIS
02B8 B7 3012  280:        STA A YAXIS       MOVE BEAM UP
02BB 08       281:        INX               MOVE TO NEXT DATA
02BC 08       282:        INX
02BD 08       283:        INX
02BE 08       284:        INX
02BF 33       285:        PUL B             COUNTER OFF STACK.
02C0 5A       286:        DEC B
02C1 26 E6    287:        BNE   DPLOT       GET NEXT POINT
02C3 BD 0389  288:        JSR   SAVED       SAVE DATA ON DISC
02C6 7F 3012  289:        CLR   YAXIS       MOVE BEAM TO AXIS
02C9 96 11    290:        LDA A XINC        AND ACROSS
02CB BB 3010  291:        ADD A XAXIS       TO RIGHT
02CE B7 3010  292:        STA A XAXIS
02D1 DE 0E    293:        LDX   XTEMP1      ANY MORE LINES ?
02D3 09       294:        DEX
02D4 26 BC    295:        BNE   PLOT1
02D6 39       296:        RTS
              297: ********************************************
              298: * PRINT TIME OF DAY:
              299: ********************************************
```



```
02D7 CE 0232  300: PTIME  LDX   #TMES
02DA BD 023D  301:        JSR   PRINT
02DD CE 0009  302:        LDX   #HOURS
02E0 BD E0BF  303:        JSR   OUT2H
02E3 86 3A    304:        LDA A #':
02E5 BD E1D1  305:        JSR   OUTCH
02E8 CE 000A  306:        LDX   #MINS
02EB BD E0BF  307:        JSR   OUT2H
02EE 86 3A    308:        LDA A #':
02F0 BD E1D1  309:        JSR   OUTCH
02F3 CE 000B  310:        LDX   #SECS
02F6 BD E0BF  311:        JSR   OUT2H
02F9 39       312:        RTS
              313: ********************************************
              314: * DELAY LOOP:
              315: ********************************************
02FA 96 08    316: DELAY  LDA A DLY
02FC 01       317: DLAY1  NOP
02FD 4A       318:        DEC A
02FE 26 FC    319:        BNE   DLAY1
0300 39       320:        RTS
              321: ********************************************
              322: * REQUEST NEW PLOT:
              323: ********************************************
0301 CE 0220  324: PREQ   LDX   #PMES
0304 BD 023D  325:        JSR   PRINT
0307 BD E07B  326:        JSR   INCH
030A 81 59    327:        CMP A #'Y
030C 39       328:        RTS
              329: ********************************************
              330: * INPUT A BYTE:
              331: ********************************************
030D BD E0AA  332: BYTE   JSR   INHEX
0310 48       333:        ASL A
0311 48       334:        ASL A
0312 48       335:        ASL A
0313 48       336:        ASL A
0314 16       337:        TAB
0315 BD E0AA  338:        JSR   INHEX
0318 1B       339:        ABA
0319 39       340:        RTS
              341: ********************************************
              342: * OPEN DISC FILE FOR 'WRITE':
              343: ********************************************
031A 86 01    344: OPNFIL LDA A #1          'WRITE OPEN' CODE
031C 97 10    345:        STA A 0$0
031E BD 6E    346:        BSR   PLCTRL
0320 39       347:        RTS               FILE NOW OPEN
              348: ********************************************
              349: * CLOSE DISC FILE FOR 'WRITE':
              350: ********************************************
0321 86 03    351: CLOSFL LDA A #3          'CLOSE' CODE
0323 97 10    352:        STA A 0$0
0325 8D 67    353:        BSR   PLCTRL
0327 39       354:        RTS
              355: ********************************************
              356: * SAVE DATA ON DISC:
              357: ********************************************
0328 86 02    358: SAVED  LDA A #2          'WRITE TO DISC' CODE
032A 97 10    359:        STA A 0$0
```

```
0332 D6 02      360:        LDA B N2
0334 D7 13      361:        STA B M
0336 CE 0426    362:        LDX   #DBASE
0339 DF 0C      363: SAVE1  STX   XTEMP
033B A6 00      364:        LDA A X
033D BD 033E    365:        JSR   FLCTRL
0340 DE 0C      366:        LDX   XTEMP
0342 08         367:        INX
0343 08         368:        INX
0344 08         369:        INX
0345 08         370:        INX
0346 7A 0013    371:        DEC   M
0349 26 EE      372:        BNE   SAVE1
034B 39         373:        RTS
                374: ********************************************
                375: *  COLLECT N SAMPLES FROM A-D
                376: ********************************************
034C A6 00      377: SAMPLE LDA A N
034E D6 01      378:        LDA B N+1
0350 36         379: SMPL1  PSH A               SAVE COUNT
0351 37         380:        PSH B
0352 86 34      381:        LDA A #$34          ZERO A/D
0354 B7 801B    382:        STA A INDATA+1
0357 86 3C      383:        LDA A #$3C          'CONVERT' COMMAND
0359 B7 801B    384:        STA A INDATA+1
035C D6 07      385:        LDA B PERIOD
035E 5A         386: SMPL2  DEC B               WAIT LOOP
035F 26 FD      387:        BNE   SMPL2
0361 5F         388:        CLR B
0362 96 03      389:        LDA A R
0364 B7 062F    390:        STA A TEMP
0367 B6 801A    391:        LDA A INDATA        GET DATA
036A 88 80      392:        EOR A #$80          2'S COMP
036C 47         393: SMPL4  ASR A
036D 56         394:        ROR B               SCALE DATA
036E 7A 062F    395:        DEC   TEMP
0371 26 F9      396:        BNE   SMPL4
0373 A7 00      397:        STA A X             SAVE DATA
0375 E7 01      398:        STA B 1,X
0377 08         399:        INX
0378 08         400:        INX
0379 32         401:        PUL A
037A 33         402:        PUL B
037B BD 0386    403:        JSR   DPDEC
037E 27 D0      404:        BEQ   SMPL1
0380 39         405:        RTS
                406: ********************************************
                407: * SET X,Y, AND Z AXES TO ZERO
                408: ********************************************
0381 4F         409: ORIGIN CLR A
0382 B7 8010    410:        STA A XAXIS
0385 B7 8012    411:        STA A YAXIS
0388 B7 8013    412:        STA A ZAXIS
038B 39         413:        RTS
                414: ********************************************
                415: * DECREMENT A DOUBLE PRECISION NUMBER
                416: ********************************************
038C C0 01      417: DPDEC  SUB B #1
038E 82 00      418:        SBC A #0
0390 26 01      419:        BNE   DPD1
```



```
0392 5D         420:        TST B
0393 39         421: DPD1   RTS
                422: ********************************************
                423: * DISC FILE MANAGEMENT SUBROUTINE
                424: ********************************************
0394 CE 063F    425: FLCTRL LDX   #FCB          CONTROL BLOCK ADDRESS
0397 D6 10      426:        LDA B #10           DFM FUNCTION CODE
0399 E7 00      427:        STA B X
039B BD 7786    428:        JSR   DFM           CALL DFM
039E 27 09      429:        BEQ   OK            NO DISC ERRORS ?
03A0 BD 72A9    430:        JSR   ZTYPDE        TYPE ERROR
03A3 BD 7783    431:        JSR   CDFM          CLOSE ACTIVE FILES
03A6 7E 7283    432:        JMP   ZWARMS        BOS
03A9 39         433: OK     RTS
                434: ********************************************
                435: * MULTIPLY INPUT DATA BY TIME WINDOW
                436: ********************************************
03AA C6 01      437: WINDOW LDA B #1
03AC 96 03      438:        LDA A R
03AE 81 08      439: JMPLN  CMP A #8
03B0 27 04      440:        BEQ   WMUL
03B2 58         441:        ASL B
03B3 4C         442:        INC A
03B4 20 F8      443:        BRA   JMPLN
03B6 58         444: WMUL   ASL B               FIND 2*256/2^R
03B7 D7 13      445:        STA B M
03B9 96 00      446:        LDA A N
03BB D6 01      447:        LDA B N+1
03BD CE 0426    448:        LDX   #DBASE
03C0 DF 1E      449:        STX   BINDX         DATA POINTER
03C2 37         450: WLOOP  PSH B               SAVE COUNT
03C3 36         451:        PSH A
03C4 DF 1C      452:        STX   WLKUP
03C6 EE 00      453:        LDX   X             GET LOOKUP
03C8 FF 0626    454:        STX   Y
03CB DE 1E      455:        LDX   BINDX
03CD EE 00      456:        LDX   X             GET DATA
03CF FF 0628    457:        STX   Z
03D2 BD 04F4    458:        JSR   MULT16        MULTIPLY TOGETHER
03D5 B6 062B    459:        LDA A P+1
03D8 F6 062C    460:        LDA B P+2           RESULT
03DB DE 1E      461:        LDX   BINDX
03DD A7 00      462:        STA A X             RESTORE DATA
03DF E7 01      463:        STA B 1,X
03E1 08         464:        INX
03E2 08         465:        INX                 MOVE TO NEXT POINT
03E3 08         466:        INX
03E4 08         467:        INX
03E5 DF 1E      468:        STX   BINDX
03E7 D6 13      469:        LDA B M
03E9 4F         470:        CLR A
03EA DB 1D      471:        ADD B WLKUP+1       FIND ADDRESS OF
03EC 99 1C      472:        ADC A WLKUP         NEXT LOOKUP
03EE D7 1D      473:        STA B WLKUP+1
03F0 97 1C      474:        STA A WLKUP
03F2 DE 1C      475:        LDX   WLKUP
03F4 32         476:        PUL A
03F5 33         477:        PUL B               GET COUNTER BACK
03F6 BD 0386    478:        JSR   DPDEC
03F9 25 C7      479:        BCS   WLOOP
```

```
03F5 39          430:          RTS
                 431: ********************************************
                 432: * SHUFFLE INPUT DATA IN BIT-REVERSED ORDER:
                 433: ********************************************
03F6 96 02       434: SHUF     LDA A N2            GET N/2
03F8 80 01       435:          SUB A #1            N/2-1
03FA 48          436:          ASL A               N-2
03FB 80 01       437:          SUB A #1            N-3
03FD 97 14       438:          STA A L             L=N-3
03FF 4F          439:          CLR A               I=0
0400 5F          440:          CLR B               J=0
0401 11          441: ILOOP    CBA                 (I-J) ( 0 ?
0402 2C 23       442:          BGE   NOSWAP        NO: DON'T SWAP
0404 37          443: SWAP     PSH B               SAVE J
0405 36          444:          PSH A               AND I
0406 4F          445:          CLR A
0407 BD 0450     446:          JSR   INDX2         GO FIND ADDRESSES
040A D7 21       447:          STA B CINDX+1
040C 97 20       448:          STA A CINDX         ADDRESS OF X(J)
040E 32          449:          PUL A               GET I
040F 36          450:          PSH A               KEEP IT
0410 16          451:          TAB
0411 4F          452:          CLR A
0412 BD 0450     453:          JSR   INDX2         FIND ADDRESS OF X(I)
0415 D7 1F       454:          STA B BINDX+1
0417 97 1E       455:          STA A BINDX
0419 DE 1E       456:          LDX   BINDX
041B EE 00       457:          LDX   0,X           GET X(I)
041D DF 0C       458:          STX   XTEMP         XTEMP=X(I)
041F DE 20       459:          LDX   CINDX
0421 A6 00       460:          LDA A X
0423 E6 01       461:          LDA B 1,X
0425 DE 1E       462:          LDX   BINDX
0427 A7 00       463:          STA A X             X(I)=X(J)
0429 E7 01       464:          STA B 1,X
042B 96 0C       465:          LDA A XTEMP
042D D6 0D       466:          LDA B XTEMP+1
042F DE 20       467:          LDX   CINDX
0431 A7 00       468:          STA A X             X(J)=XTEMP
0433 E7 01       469:          STA B 1,X
0435 32          470:          PUL A
0436 33          471:          PUL B
0437 36          472: NOSWAP   PSH A
0438 96 02       473:          LDA A N2
043A 97 16       474:          STA A K             K=N/2
043C 32          475:          PUL A
043D D1 16       476: TSTJ     CMP B K             J >= K ?
043F 2D 07       477:          BLT   NOCHNG
0441 D0 16       478: CHNG     SUB B K             J=J-K
0443 77 0016     479:          ASR   K             K=K/2
0446 20 F5       480:          BRA   TSTJ
0448 DB 16       481: NOCHNG   ADD B K             J=J+K
044A 4C          482:          INC A               I=I+1
044B 91 14       483:          CMP A L             I > N-2 ?
044D 2F B2       484:          BLE   ILOOP
044F 39          485:          RTS
                 486: *
0450 58          487: INDX2    ASL B
0451 49          488:          ROL A
0452 58          489:          ASL B
```



```
0453 49          490:          ROL A
0454 CE 0426     491:          LDX   #DBASE
0457 DF 0C       492:          STX   XTEMP
0459 DB 0D       493:          ADD B XTEMP+1
045B 99 0C       494:          ADC A XTEMP
045D 39          495:          RTS
                 496: ********************************************
                 497: *  EVALUATE SQUARE ROOT OF NUMBER IN ACCA:ACCB.
                 498: *  THIS SUBROUTINE EVALUATES THE NEWTON-RAPHSON
                 499: *  RECURSION FORMULA:
                 500: *  X0 = X0 - ((X0^2-X)/2*X0)
                 501: *     WHERE X0 IS A BETTER APPROXIMATION TO THE
                 502: *  SQUARE ROOT OF X
                 503: ********************************************
045E 7F 063A     504: SQRT     CLR   FLAG          CLEAR PRECISION INDICATOR
0461 B7 063D     505:          STA A G+2
0464 F7 063E     506:          STA B G+3
0467 7F 063B     507:          CLR   G
046A 7F 063C     508:          CLR   G+1
046D B7 0626     509: SQRT1    STA A Y             PUT IN MULTIPLIER
0470 B7 0628     510:          STA A Z             AND MULTIPLICAND
0473 F7 0627     511:          STA B Y+1
0476 F7 0629     512:          STA B Z+1
0479 BD 04F4     513:          JSR   MULT16
047C B6 062C     514:          LDA A P+2
047F F6 062D     515:          LDA B P+3
0482 F0 063E     516:          SUB B G+3
0485 B2 063D     517:          SBC A G+2
0488 F7 0635     518:          STA B XKDVND+3      RESULT IN DIVIDEND
048B B7 0634     519:          STA A XKDVND+2
048E F6 0628     520:          LDA B P+1
0491 B6 062A     521:          LDA A P
0494 F2 063C     522:          SBC B G+1
0497 B2 063B     523:          SBC A G
049A F7 0633     524:          STA B XKDVND+1
049D B7 0632     525:          STA A XKDVND
04A0 F6 0629     526:          LDA B Z+1
04A3 B6 0628     527:          LDA A Z
04A6 58          528:          ASL B               FIND 2*X0
04A7 49          529:          ROL A
04A8 B7 0630     530:          STA A XKDVSR        PUT IN DIVISOR
04AB F7 0631     531:          STA B XKDVSR+1
04AE BD 0553     532:          JSR   DIVIDE        FIND ((X0*X0-X)/2*X0)
04B1 B6 0628     533:          LDA A Z
04B4 F6 0629     534:          LDA B Z+1
04B7 F0 0639     535:          SUB B XKQUOT+3
04BA B2 0638     536:          SBC A XKQUOT+2      X0-((X0*X0-X)/2*X0)
04BD B1 0628     537:          CMP A Z             MSBYTE SAME ?
04C0 26 AB       538:          BNE   SQRT1         NO: ITERATE AGAIN
04C2 36          539:          PSH A               SAVE MSBYTE
04C3 B6 0629     540:          LDA A Z+1
04C6 10          541:          SBA                 X0(OLD)-X0(NEW)
04C7 81 01       542:          CMP A #1            IF DIFFERENT BY 1
04C9 27 08       543:          BEQ   SQRT2         OR EQUAL THEN TEST
04CB 81 00       544:          CMP A #0            FOR PRECISION LEVEL
04CD 27 04       545:          BEQ   SQRT2
04CF 32          546:          PUL A
04D0 7E 046D     547:          JMP   SQRT1
04D3 32          548: SQRT2    PUL A               RESTORE MSBYTE
04D4 7D 063A     549:          TST   FLAG
```

```
04D7 26 1F     600:        BNE   DONE
04D9 7C 0639   601:        INC   FLAG
04DC B6 063E   602:        LDA A G+3
04DF B7 0631   603:        STA A G+1
04E2 B6 063D   604:        LDA A G+2
04E5 B7 0630   605:        STA A G
04E8 7F 063D   606:        CLR   G+2
04EB 7F 063E   607:        CLR   G+3
04EE 17        608:        TBA
04EF 5F        609:        CLR B
04F0 7E 0460   610:        JMP   SQRT1
04F3 39        611: DONE   RTS
               612: ********************************************
               613: * SIGNED MULTIPLY ROUTINE:
               614: *  MULTIPLIES 2 16-BIT 2'S COMPLEMENT NUMBERS
               615: * USING THE BOOTH ALGORITHM.
               616: *
               617: * MULTIPLIER: Y,Y+1
               618: * MULTIPLICAND: Z,Z+1
               619: * PRODUCT: P,P+1,P+2,P+3
               620: *
               621: * THE MULTIPLIER IS DESTROYED
               622: * THE MULTIPLICAND IS UNCHANGED
               623: *
               624: ********************************************
04F4 CE 0000   625: MULT16 LDX   #0
04F7 FF 0624   626:        STX   P              CLEAR REGISTERS
04FA FF 0626   627:        STX   P+2
04FD 7F 063C   628:        CLR   TB
0500 CE 0010   629:        LDX   #16
0503 96 0627   630: LOOP   LDA A Y+1
0505 84 01     631:        AND A #1
0507 16        632:        TAB
0508 B8 063C   633:        EOR A TB             Y(LSB)=Y(LSB-1) ?
050B 27 29     634:        BEQ   SHIFT          YES: SHIFT
050D 5D        635:        TST B
050E 27 14     636:        BEQ   ADD            Y(LSB)=0: ADD.
0510 96 0624   637: SUBTR  LDA A P
0512 B6 0628   638:        LDA A P+1
0515 F0 0629   639:        SUB B Z+1
0518 B2 0628   640:        SBC A Z
051B B7 0624   641:        STA A P
051E F7 0628   642:        STA B P+1
0521 20 12     643:        BRA   SHIFT
0523 96 0624   644: ADD    LDA A P
0525 F6 0628   645:        LDA B P+1
0528 FB 0629   646:        ADD B Z+1
052B 89 0628   647:        ADC A Z
052D B7 0624   648:        STA A P
0530 F7 0628   649:        STA B P+1
0533 78 063C   650: SHIFT  CLR   TB
0536 76 0626   651:        ROR   Y
0539 76 0627   652:        ROR   Y+1
053C 78 063C   653:        ROL   TB
053F 76 0624   654:        ROR   P
0542 76 0625   655:        ROR   P+1
0545 76 0626   656:        ROR   P+2
0548 76 0627   657:        ROR   P+3
054B 09        658:        DEX
054C 26 B1     659:        BNE   LOOP
```



```
0552 39        660:        RTS
               661: ********************************************
               662: * DIVISION ROUTINE.
               663: *  DIVIDES A 32-BIT 2'S COMPLEMENT NUMBER BY A
               664: * 16-BIT 2'S COMPLEMENT NUMBER TO GIVE A 16-BIT
               665: * RESULT.
               666: *
               667: * DIVIDEND: XKDVND,XKDVND+1,XKDVND+2,XKDVND+3
               668: * DIVISOR:  XKDVSR,XKDVSR+1
               669: * QUOTIENT: XKQUOT,XKQUOT+1,XKQUOT+2,XKQUOT+3
               670: *
               671: * FROM "MOTOROLA APPLICATIONS MANUAL". PP. 2.23-24.
               672: *
               673: ********************************************
0553           674: DIVIDE EQU   *
0553 C6 10     675: XKDIVD LDA B #16          S=16
0555 7F 0636   676:        CLR   XKQUOT
0558 7F 0637   677:        CLR   XKQUOT+1
055B 7F 0638   678:        CLR   XKQUOT+2     CLEAR QUOTIENT BUFFER
055E 7F 0639   679:        CLR   XKQUOT+3
0561 5C        680: DVDLP0 INC B              S=S+1
0562 C1 20     681:        CMP B #32
0564 2E 61     682:        BGT   DVDERR       IF S>32 ERROR
0566 78 0631   683:        ASL   XKDVSR+1     LEFT SHIFT DIVISOR
0569 79 0630   684:        ROL   XKDVSR
056C 24 F3     685:        BCC   DVDLP0       IF C=0 DON'T LOOP
056E 76 0630   686:        ROR   XKDVSR       SHIFT DIVISOR BACK 1
0571 76 0631   687:        ROR   XKDVSR+1
0574 37        688:        PSH B              SHIFT COUNT ON STACK
0575 B6 0632   689:        LDA A XKDVND
0578 B1 0630   690: DVDLP1 CMP A XKDVSR       IF DIVIDEND<DIVISOR
057B 25 21     691:        BCS   DVNSUB       DON'T SUBTRACT
057D 0D        692: DVDLP2 SEC                OF DIVIDEND> OR =
057E 79 0639   693:        ROL   XKQUOT+3     DIVISOR THEN
0581 79 0638   694:        ROL   XKQUOT+2     SHIFT 0 LEFT
0584 79 0637   695:        ROL   XKQUOT+1     1 BIT.
0587 79 0636   696:        ROL   XKQUOT
058A B6 0632   697:        LDA A XKDVND
058D F6 0633   698:        LDA B XKDVND+1     Y(M)=Y(M)-X
0590 F0 0631   699:        SUB B XKDVSR+1
0593 B2 0630   700:        SBC A XKDVSR
0596 F7 0633   701:        STA B XKDVND+1
0599 B7 0632   702:        STA A XKDVND
059C 20 0D     703:        BRA   DVSHFT
059E 0C        704: DVNSUB CLC                SHIFT Q LEFT
059F 79 0639   705:        ROL   XKQUOT+3
05A2 79 0638   706:        ROL   XKQUOT+2
05A5 79 0637   707:        ROL   XKQUOT+1
05A8 79 0636   708:        ROL   XKQUOT
05AB 33        709: DVSHFT PUL B
05AC 5A        710:        DEC B
05AD 37        711:        PSH B
05AE 27 1A     712:        BEQ   DVDEND
05B0 0C        713:        CLC                IF S>0 SHIFT DIVIDEND
05B1 79 0635   714:        ROL   XKDVND+3
05B4 79 0634   715:        ROL   XKDVND+2
05B7 79 0633   716:        ROL   XKDVND+1
05BA 79 0632   717:        ROL   XKDVND
05BD B6 0632   718:        LDA A XKDVND
05C0 F6 0633   719:        LDA B XKDVND+1
```

```
0503 25 B8     720:          BCS   DVDLP2       IF C=1 GOTO LOOP2
0505 20 B1     721:          BRA   DVDLP1
0507 7E E0E3   722: DVDERR   JMP   CNTRL
050A 33        723: DVDEND   PUL B
050B 39        724:          RTS
               725: ********************************************
               726: *      INTERRUPT ROUTINE
               727: * UPDATES REAL TIME CLOCK
               728: ********************************************
050C 96 08     729: CLK      LDA A SECS
050E 8B 01     730:          ADD A #1
0510 19        731:          DAA
0511 97 08     732:          STA A SECS
0513 81 60     733:          CMP A #$60
0515 26 1C     734:          BNE   CK1
0517 7F 0008   735:          CLR   SECS
051A 96 09     736:          LDA A MINS
051C 8B 01     737:          ADD A #1
051E 19        738:          DAA
051F 97 09     739:          STA A MINS
0521 81 60     740:          CMP A #$60
0523 26 0E     741:          BNE   CK1
0525 96 0A     742:          LDA A HOURS
0527 8B 01     743:          ADD A #1
0529 19        744:          DAA
052A 97 0A     745:          STA A HOURS
052C 81 24     746:          CMP A #$24
052E 26 03     747:          BNE   CK1
0530 7F 0009   748:          CLR   HOURS
0533 B6 8003   749: CK1      LDA A CLOCK
0536 3B        750:          RTI
               751: ********************************************
               752: * COMPUTE POWER SPECTRUM
               753: *  P=SQR(RE*RE+IM*IM)
               754: ********************************************
0537 CE 0426   755: PSPEC    LDX   #DBASE
053A DF 0C     756:          STX   XTEMP
053C D6 02     757:          LDA B N2
053E 37        758: POWER1   PSH B                SAVE COUNT
053F DE 0C     759:          LDX   XTEMP
0541 EE 00     760:          LDX   0,X            GET REAL DATA
0543 FF 0626   761:          STX   Y
0546 FF 0623   762:          STX   Z
0549 BD 04F4   763:          JSR   MULT16         FIND RE*RE
054C EE 02     764:          LDX   2,X
054E FF 0626   765:          STX   Y
0551 FF 0623   766:          STX   Z
0554 96 062A   767:          LDA A P
0557 F6 062B   768:          LDA B P+1
055A 37        769:          PSH B
055B 36        770:          PSH A
055C BD 04F4   771:          JSR   MULT16
055F 32        772:          PUL A
0560 33        773:          PUL B
0561 FB 062B   774:          ADD B P+1            R*R+I*I
0564 B9 062A   775:          ADC A P
0567 C1 00     776:          CMP B #0
0569 26 06     777:          BNE   SQROOT
056B 81 00     778:          CMP A #0
056D 26 02     779:          BNE   SQROOT
```



```
062F 20 03     780:          BRA   HOP
0631 BD 045E   781: SQROOT   JSR   SQRT
0634 DE 0C     782: HOP      LDX   XTEMP
0636 A7 00     783:          STA A X
0638 E7 01     784:          STA B 1,X
063A 96 0C     785: NXLOC    LDA A XTEMP
063C D6 0D     786:          LDA B XTEMP+1
063E CB 04     787:          ADD B #4
0640 89 00     788:          ADC A #0
0642 97 0C     789:          STA A XTEMP
0644 D7 0D     790:          STA B XTEMP+1        NEW POINTER
0646 33        791:          PUL B
0647 5A        792:          DEC B
0648 26 B4     793:          BNE   POWER1
064A 39        794:          RTS
               795: *
               796: * FFT ROUTINES:
               797: *
064B           798: FFT      EQU   *
               799:          END
     NO ERROR(S) DETECTED
```

DOS:

listings for chapter 4

(1) Memory diagnostic

(2) Vector diagnostic

(3) Parallel processing diagnostic

```
                        NAM     CDAT1
                    *MEM DIAGNOSTIC (JOHN CHRISTENSEN'S)
                        OPT     O
     E0E3       CONTRL EQU     $E0E3
A002                    ORG     $A002
A002 0002       LOTEMP RMB     2           STARTING ADDRESS
A004 0002       HITEMP RMB     2           ENDING ADDRESS


A014                    ORG     $A014
A014 00         INIPAT FCB     0           INITIAL TEST PATTERN
A015 FF         TESPAT FCB     $FF         TEST PATTERN
A016 0002       IXRTMP RMB     2           IXR TEMPORARY STORAGE
A018 FE A002    START  LDX     LOTEMP
A01B B6 A014           LDA A   INIPAT
A01E A7 00      LOOP1  STA A   0,X
A020 A1 00             CMP A   0,X
A022 26 53             BNE     ERPNT1
A024 BC A004           CPX     HITEMP
A027 27 03             BEQ     LOAPAT
A029 08                INX
A02A 20 F2             BRA     LOOP1
A02C FE A002 LOAPAT    LDX     LOTEMP
A02F F6 A015           LDA B   TESPAT
A032 E7 00      LOOP4  STA B   0,X
A034 20 14             BRA     CHECK


A048                    ORG     $A048
A048 A018               FDB     $A018
A04A E1 00      CHECK  CMP B   0,X
A04C 26 2A             BNE     ERPNT2
A04E FF A016 CHKLOW    STX     IXRTMP
A051 BC A002 LOOP2     CPX     LOTEMP
A054 27 07             BEQ     CHCKHI
A056 09                DEX
A057 A1 00             CMP A   0,X
A059 26 1E             BNE     ERPNT3
A05B 20 F4             BRA     LOOP2
A05D FE A016 CHCKHI    LDX     IXRTMP
A060 BC A004           CPX     HITEMP
A063 27 16             BEQ     END
A065 08         LOOP3  INX
A066 A1 00             CMP A   0,X
A068 26 10             BNE     ERPNT4
A06A BC A004           CPX     HITEMP
A06D 26 F6             BNE     LOOP3
A06F FE A016 RESTRE    LDX     IXRTMP
A072 A7 00             STA A   0,X
A074 08                INX
A075 20 BB             BRA     LOOP4
A077 3F         ERPNT1 SWI                 ERROR ON INITIAL PATTERN
A078 3F         ERPNT2 SWI                 ERROR ON TEST PATTERN
A079 3F         ERPNT3 SWI                 DUAL ADDRESS ERROR LOW
A07A 3F         ERPNT4 SWI                 DUAL ADDRESS ERROR HI
A07B 7E E0E3    END    JMP     CONTRL
                       END
```

```
SLAVE VECTOR TEST ROUTINE              SSB MNEMONIC ASSEMBLER  PAGE 1

                        1:        NAM   SLAVE VECTOR TEST ROUTINE
                        2:        OPT   NOS,LIS,PAG
                        3: ********************************************
                        4: *          SLAVE PROCESSOR DIAGNOSTIC 2
                        5: *          ============================
                        6: *
                        7: * ROUTINE TO TEST SLAVE INTERRUPT VECTORS. AN INTERRUPT
                        8: * SEQUENCE IS GENERATED BY THE MASTER, RESULTING IN THE
                        9: * LOADING OF A TEST BYTE WITH $CC, $DD, $EE, OR $FF,
                       10: * DEPENDING ON THE TYPE OF INTERRUPT GENERATED.
                       11: *
                       12: ********************************************
0001                   13: TSTBYT EQU   $0001       TEST BYTE
C000                   14: LATCH  EQU   $C000       BASE OF SLAVE RAM
                       15: * VECTOR AREA:
C3F8                   16:        ORG   $C3F8
C3F8 01 00             17: IRQ    FDB   IRQTST-LATCH
C3FA 01 06             18: SWI    FDB   SWITST-LATCH
C3FC 01 0A             19: NMI    FDB   NMITST-LATCH
C3FE 01 0E             20: RES    FDB   RESTST-LATCH
                       21: *
C100                   22:        ORG   $C100
C100 86 CC             23: IRQTST LDA A #$CC
C102 B7 01             24: LOOP   STA A TSTBYT
C104 20 FC             25:        BRA   LOOP
C106 86 DD             26: SWITST LDA A #$DD
C108 20 F8             27:        BRA   LOOP
C10A 86 EE             28: NMITST LDA A #$EE
C10C 20 F4             29:        BRA   LOOP
C10E 86 FF             30: RESTST LDA A #$FF
C110 20 F0             31:        BRA   LOOP
                       32:        END
     NO ERROR(S) DETECTED
```

```
                    1:          NAM   PARALLEL PROCESSING TEST ROUTINE
                    2:          OPT   NOS,LIS,PAG
                    3: ********************************************
                    4: *
                    5: *          SLAVE PROCESSOR DIAGNOSTIC 3
                    6: *          ============================
                    7: *
                    8: * ROUTINE TO DEMONSTRATE THE PARALLEL PROCESSING
                    9: * CAPABILITY OF THE MASTER-SLAVE SYSTEM. THE
                   10: * EXPRESSION (A1 X B1) + (C1 X D1) IS EVALUATED
                   11: * BY EXECUTING THE MULTIPLICATIONS IN PARALLEL.
                   12: * ALL OPERANDS ARE 2-BYTE 2'S COMPLEMENT NUMBERS.
                   13: * RESULT IN "P".
                   14: *
                   15: ********************************************
                   16: *
                   17: * MASTER VARIABLE AREA:
                   18: *
0000               19:          ORG   0
0000               20: A1       RMB   2           OPERANDS
0002               21: B1       RMB   2
0004               22: C1       RMB   2
0006               23: D1       RMB   2
0008               24: P        RMB   4           RESULT
000C               25: Y1       RMB   2           MULTIPLIER
000E               26: Z1       RMB   2           MULTIPLICAND
0010               27: P1       RMB   4           PRODUCT
0014               28: TB1      RMB   1           TEST BYTE
                   29: *
                   30: * SLAVE VARIABLE AREA:
                   31: *
C000               32: LATCH    EQU   $C000
0001               33:          ORG   $0001
0001               34: TSTBYT   RMB   1
0002               35: TB2      RMB   1           TEST BYTES
0003               36: Y2       RMB   2           MULTIPLIER
0005               37: Z2       RMB   2           MULTIPLICAND
0007               38: P2       RMB   4           PRODUCT
                   39: *
                   40: * MASTER PROCESSOR ROUTINES:
                   41: *
0100               42:          ORG   $0100
0100 86 FE         43: BEGIN    LDA A #$FE
0102 B7 C000       44:          STA A LATCH       HOLD SLAVE IN RESET
0105 DE 00         45:          LDX   A1          GET OPERANDS
0107 FF C003       46:          STX   Y2+LATCH    INTO SLAVE RAM
0109 DE 02         47:          LDX   B1
010B FF C005       48:          STX   Z2+LATCH
010E 86 FF         49:          LDA A #$FF
0110 B7 C000       50:          STA A LATCH       START SLAVE.
0113 DE 04         51:          LDX   C1          GET OPERANDS
0115 DF 0C         52:          STX   Y1          INTO RAM
0117 DE 06         53:          LDX   D1
0119 DF 0E         54:          STX   Z1
011B BD 0141       55:          JSR   MULT1       FIND (C X D)
011E 7D 0001       56: LOOP     TST   TSTBYT      SLAVE FINISHED ?
0121 26 FB         57:          BNE   LOOP        IF NOT: WAIT.
0123 D6 13         58:          LDA B P1+3        FIND SUM OF PRODUCTS
0125 96 12         59:          LDA A P1+2
```



```
0127 FB C00A       60:          ADD B P2+LATCH+3
012A B9 C009       61:          ADC A P2+LATCH+2
012D 97 0A         62:          STA A P+2         SAVE PARTIAL RESULT
012F D7 0B         63:          STA B P+3
0131 D6 11         64:          LDA B P1+1
0133 96 10         65:          LDA A P1
0135 F9 C008       66:          ADC B P2+LATCH+1
0138 B9 C007       67:          ADC A P2+LATCH
013B 97 08         68:          STA A P           SAVE FULL RESULT
013D D7 09         69:          STA B P+1
013F 3F            70:          SWI               INTERRUPT TO FINISH
                   71: *
                   72: * 2'S COMPLEMENT MULTIPLY ROUTINE:
                   73: *
0140 CE 0000       74: MULT1    LDX   #0          CLEAR REGISTERS
0143 DF 10         75:          STX   P1
0145 DF 12         76:          STX   P1+2
0147 7F 0014       77:          CLR   TB1
014A CE 0010       78:          LDX   #16
014D 96 0D         79: LOOP1    LDA A Y1+1
014F 84 01         80:          AND A #1
0151 16            81:          TAB
0152 98 14         82:          EOR A TB1
0154 27 1D         83:          BEQ   SHIFT1
0156 5D            84:          TST B
0157 27 0E         85:          BEQ   ADD1
0159 96 10         86: SUB1     LDA A P1
015B D6 11         87:          LDA B P1+1
015D D0 0F         88:          SUB B Z1+1
015F 92 0E         89:          SBC A Z1
0161 97 10         90:          STA A P1
0163 D7 11         91:          STA B P1+1
0165 20 0C         92:          BRA   SHIFT1
0167 96 10         93: ADD1     LDA A P1
0169 D6 11         94:          LDA B P1+1
016B DB 0F         95:          ADD B Z1+1
016D 99 0E         96:          ADC A Z1
016F 97 10         97:          STA A P1
0171 D7 11         98:          STA B P1+1
0173 7F 0014       99: SHIFT1   CLR   TB1
0176 76 000C      100:          ROR   Y1
0179 76 000D      101:          ROR   Y1+1
017C 79 0014      102:          ROL   TB1
017F 77 0010      103:          ASR   P1
0182 76 0011      104:          ROR   P1+1
0185 76 0012      105:          ROR   P1+2
0188 76 0013      106:          ROR   P1+3
018B 09           107:          DEX
018C 26 BF        108:          BNE   LOOP1
018E 78 0012      109:          ASL   P1+2
0191 79 0011      110:          ROL   P1+1
0194 79 0010      111:          ROL   P1
0197 39           112:          RTS
                  113: *
                  114: * SLAVE PROCESSOR ROUTINES
                  115: *
C100              116:          ORG   $C100
C100 8E 03E0      117: SLAVE    LDS   #$03E0      SET UP STACK
C103 86 FF        118:          LDA A #$FF        SET FLAG
C105 97 01        119:          STA A TSTBYT
```

```
C107 8D 05    120:         BSR   MULT2
C109 7F 0001  121: WAIT    CLR   TSTBYT
C10C 20 FB    122:         BRA   WAIT
              123: *
              124: * 2'S COMPLEMENT MULTIPLY ROUTINE:
              125: *
C10E CE 0000  126: MULT2   LDX   #0
C111 DF 07    127:         STX   P2          CLEAR REGISTERS
C113 DF 09    128:         STX   P2+2
C115 7F 0002  129:         CLR   TB2
C118 CE 0010  130:         LDX   #16         SHIFT COUNTER
C11B 96 04    131: LOOP2   LDA A Y2+1
C11D 84 01    132:         AND A #1
C11F 16       133:         TAB
C120 98 02    134:         EOR A TB2
C122 27 1D    135:         BEQ   SHIFT2
C124 5D       136:         TST B
C125 27 0E    137:         BEQ   ADD2
C127 96 07    138: SUB2    LDA A P2
C129 D6 08    139:         LDA B P2+1
C12B D0 06    140:         SUB B Z2+1
C12D 92 05    141:         SBC A Z2
C12F 97 07    142:         STA A P2
C131 D7 08    143:         STA B P2+1
C133 20 0C    144:         BRA   SHIFT2
C135 96 07    145: ADD2    LDA A P2
C137 D6 08    146:         LDA B P2+1
C139 DB 06    147:         ADD B Z2+1
C13B 99 05    148:         ADC A Z2
C13D 97 07    149:         STA A P2
C13F D7 08    150:         STA B P2+1
C141 7F 0002  151: SHIFT2  CLR   TB2
C144 76 0003  152:         ROR   Y2
C147 76 0004  153:         ROR   Y2+1
C14A 79 0002  154:         ROL   TB2
C14D 77 0007  155:         ASR   P2
C150 76 0008  156:         ROR   P2+1
C153 76 0009  157:         ROR   P2+2
C156 76 000A  158:         ROR   P2+3
C159 09       159:         DEX
C15A 26 BF    160:         BNE   LOOP2
C15C 78 0009  161:         ASL   P2+2
C15F 79 0003  162:         ROL   P2+1
C162 79 0007  163:         ROL   P2
C165 39       164:         RTS
              165: *
              166: * SLAVE RESET VECTOR
              167: *
03F8          168:         ORG   $03F8
03F8 01 00    169:         FDB   SLAVE-LATCH
              170:         END
     NO ERROR(S) DETECTED
```

listings for chapter 5

(1) BCH Encoder

(2) BCH Decoder

```
(15,7) BCH ENCODER                    6809 MNEMONIC ASSEMBLER  PAGE 1

                        1:         NAM   (15,7) BCH ENCODER
                        2:         OPT   MOS,LIS
                        3: ********************************************
                        4: *
                        5: *        (15,7) BCH ENCODING SUBROUTINE
                        6: *        ==============================
                        7: *
                        8: * SUBROUTINE TO MAP A 7-BIT INFORMATION BLOCK INTO
                        9: * A (15,7) CODEWORD.
                       10: * DATA BLOCK IS DIVIDED BY THE GENERATOR POLYNOMIAL
                       11: * TO FIND CHECK BITS. DATA IN ACCA. CHECK BITS
                       12: * RETURNED IN ACCB.
                       13: *
                       14: *
                       15: ********************************************
                       16: *
                       17: *  DATA STORAGE AREA:
                       18: *
0000                   19:         ORG   0
0000 98                20: GENPOL  FCB   %10011000   M1(X)
0001 F8                21:         FCB   %11111000   M3(X)
0002                   22: SHFCNT  RMB   1
0100                   23:         ORG   $0100
0100 36                24: CODE    PSH A
0101 C6 08             25: CODE1   LDA B #8
0103 D7 02             26:         STA B SHFCNT
0105 5F                27:         CLR B
0106 D8 01             28: EXCROR  EOR B GENPOL+1
0108 98 00             29:         EOR A GENPOL
010A 85 80             30: BITTST  BIT A #$80
010C 26 F8             31:         BNE   EXCROR
010E 59                32:         ROL B
010F 49                33:         ROL A
0110 7A 0002           34:         DEC   SHFCNT
0113 26 F5             35:         BNE   BITTST
0115 16                36: CODE2   TAB
0116 32                37:         PUL A
0117 39                38:         RTS
                       39:         END
     NO ERROR(S) DETECTED



000:
```

```
                    1:         NAM   (15,7) BCH DECODER
                    2:         OPT   NOS,LIS
                    3: ********************************************
                    4: *
                    5: *       (15,7) BCH DECODING SUBROUTINE
                    6: *       ==============================
                    7: *
                    8: * SUBROUTINE TO CORRECT UP TO 2 ERRORS IN A
                    9: * 15-BIT RECEIVED WORD. RECEIVED WORD
                   10: * CONTAINED IN RX, RX+1
                   11: *
                   12: ********************************************
                   13: *
                   14: * VARIABLE STORAGE AREA:
                   15: *
0000               16:         ORG   0
0000               17: RX      RMB   2           RECEIVED WORD R(X)
0002 93            18: M1      FCB   %10011000   MINIMUM POLYNOMIAL M1(X)
0003 F8            19: M3      FCB   %11111000   MINIMUM POLYNOMIAL M3(X)
0004               20: M       RMB   1           DIVISOR
0005               21: TB1PTR  RMB   2           LOOKUP TABLE POINTERS
0007               22: TB2PTR  RMB   2
0009               23: S1      RMB   1           POWER SUM SYMMETRIC FUNCTION S1
000A               24: S3      RMB   1           POWER SUM SYMMETRIC FUNCTION S3
000B               25: SIGMA2  RMB   1           ELEMENTARY SYMMETRIC FUNCTION 2
000C               26: SHFCNT  RMB   1           SHIFT COUNTER
000D               27: S1EXP   RMB   1           EXPONENT OF S1
000E               28: TEMP1   RMB   1           TEMPORARY DATA STORE
000F               29: I       RMB   1           GALOIS FIELD EXPONENT COUNT
0010 E8            30: GENPOL  FCB   %11101000,%10000000
                   31: *
0300               32:         ORG   $0300       EXPONENTS >> FIELD ELEMENTS
0300 01            33: TBL1    FCB   $1,$2,$4,$8,$3,$6,$C
0307 09            34:         FCB   $B,$5,$A,$7,$E,$F,$D,$9,$1
                   35: *
0400               36:         ORG   $0400       FIELD ELEMENTS >> EXPONENTS
0400 00            37: TBL2    FCB   0,$0,$1,$4,$2,$8,$5,$A,$3
0409 0E            38:         FCB   $E,$9,$7,$6,$D,$B,$C
                   39: ********************************************
                   40: *  BEGIN DECODING PROCEDURE                *
                   41: ********************************************
0100               42:         ORG   $0100
0100 CE 0300       43: DECODE  LDX   #TBL1
0103 DF 05         44:         STX   TB1PTR      INITIALISE LOOKUP POINTERS
0105 CE 0400       45:         LDX   #TBL2
0108 DF 07         46:         STX   TB2PTR
                   47: ********************************************************
                   48: *  FIND POWER SUM SYMMETRIC FUNCTIONS S1 & S3            *
                   49: ********************************************************
                   50: *
                   51: * DIVISION ROUTINE TO FIND RESIDUE
                   52: * OF R(X)/M :
                   53: *
010A C6 0F         54: SYND    LDA B #15         INITIALISE SHIFT COUNT
010C D7 0C         55:         STA B SHFCNT
010E 96 00         56:         LDA A RX          GET RECEIVED WORD MSB
0110 D6 01         57:         LDA B RX+1        & LSB
0112 88 03         58: DIV1    EOR A M1
0114 25 30         59: TST1    BIT A #$80        LEFT JUSTIFIED ?
```



```
0116 26 FA         60:         BNE   DIV1       YES: EXOR AGAIN
0118 58            61:         ASL B             NO: SHIFT DIVIDEND LEFT
0119 49            62:         ROL A
011A 7A 000C       63:         DEC   SHFCNT
011D 26 F5         64:         BNE   TST1
011F 44            65:         LSR A             RIGHT JUSTIFY REMAINDER
0120 44            66:         LSR A
0121 44            67:         LSR A
0122 44            68:         LSR A
0123 97 09         69:         STA A S1
                   70: ********************************************
                   71: * COMPUTE S3 BY USING LOOKUP TABLES TO SUM  *
                   72: * POWERS OF R(ALPHA**3).                    *
                   73: ********************************************
0125 7F 000A       74: SYND3   CLR   S3          ZERO S3
0128 C6 0E         75:         LDA B #14         EXPONENT COUNT
012A D7 0C         76:         STA B SHFCNT
012C 96 00         77:         LDA A RX          GET WORD
012E D6 01         78:         LDA B RX+1
0130 85 80         79: SYND30  BIT A #$80        LEFT JUSTIFIED ?
0132 26 12         80:         BNE   BIT         YES: FIND FIELD ELEMENT
0134 58            81: SYND31  ASL B             SHIFT WORD
0135 49            82:         ROL A
0136 7A 000C       83:         DEC   SHFCNT      NEXT EXPONENT
0139 2C F5         84:         BGE   SYND30      BRANCH IF >=0
013B 7D 0009       85:         TST   S1          S1=0?
013E 26 23         86:         BNE   SIG2
0140 7D 000A       87:         TST   S3          S3=0?
0143 26 1E         88:         BNE   SIG2
0145 39            89: NOERR   RTS               ERROR-FREE WORD
0146 37            90: BIT     PSH B             SAVE ACCS ON STACK
0147 36            91:         PSH A
0148 D6 0C         92:         LDA B SHFCNT      GET CURRENT EXPONENT
014A 17            93:         TBA
014B 48            94:         ASL A             MULTIPLY BY 3
014C 1B            95:         ABA
014D 81 0F         96: SYND32  CMP A #15         ADD MODULO-15
014F 2D 04         97:         BLT   SYND33
0151 80 0F         98:         SUB A #15
0153 20 F8         99:         BRA   SYND32      GO TEST AGAIN
0155 97 06        100: SYND33  STA A TB1PTR+1    SAVE EXP IN PTR
0157 DE 05        101:         LDX   TB1PTR      GET POINTER
0159 A6 00        102:         LDA A X           GET FIELD ELEMENT
015B 98 09        103:         EOR A S3          ADD IN TO S3
015D 97 0A        104:         STA A S3          AND RETURN NEW VALUE
015F 32           105:         PUL A             RESTORE STACK
0160 33           106:         PUL B
0161 20 D1        107:         BRA   SYND31
                  108: ********************************************
                  109: * FIND SIGMA2=S1*S1+S3/S1  (GF(2**4))       *
                  110: ********************************************
0163 96 09        111: SIG2    LDA A S1          GET S1
0165 8D 10        112:         BSR   GFSQR       FIND S1*S1
0167 97 0E        113:         STA A TEMP1       SAVE RESULT
0169 96 0A        114:         LDA A S3
016B 27 04        115:         BEQ   ZERO        BRANCH IF ZERO
016D D6 09        116:         LDA B S1
016F 8D 1C        117:         BSR   GFDIV       FIND S3/S1
0171 98 0E        118: ZERO    EOR A TEMP1       S1*S1+S3/S1
0173 97 0B        119:         STA A SIGMA2
```

```
0175 20 2E     120:         BRA   CONT        CONTINUE WITH MAIN PROCEDURE
               121: ********************************************
               122: * GALOIS FIELD SQUARING ROUTINE:           *
               123: *  FINDS ACCA=ACCA*ACCA IN GF(2**4)        *
               124: ********************************************
0177 97 09     125: GFSQR   STA A TB2PTR+1    FIELD ELEMENT >>POINTER
0179 DE 07     126:         LDX   TB2PTR      FETCH POINTER
017B A6 00     127:         LDA A X           FETCH EXPONENT OF ROOT
017D 97 0D     128:         STA A S1EXP       SAVE EXPONENT
017F 48        129:         ASL A             MULTIPLY BY 2
0180 81 0F     130:         CMP A #15         MODULO-15
0182 2D 02     131:         BLT   SQ1
0184 80 0F     132:         SUB A #15
0186 97 06     133: SQ1     STA A TB1PTR+1    NEW EXPONENT >>POINTER
0188 DE 05     134:         LDX   TB1PTR      FETCH POINTER
018A A6 00     135:         LDA A X           FETCH NEW FIELD ELEMENT
018C 39        136:         RTS
               137: ********************************************
               138: * GALOIS FIELD DIVISION ROUTINE:           *
               139: *   FINDS ACCA = ACCA/ACCB IN GF(2**4)     *
               140: ********************************************
018D 97 09     141: GFDIV   STA A TB2PTR+1    FIELD ELEMENT >>POINTER
018F DE 07     142:         LDX   TB2PTR      FETCH POINTER
0191 A6 00     143:         LDA A X           FETCH EXPONENT
0193 D7 09     144:         STA B TB2PTR+1    DIVISOR FIELD ELEMENT
0195 DE 07     145:         LDX   TB2PTR
0197 E6 00     146:         LDA B X           DIVISOR EXPONENT
0199 10        147:         SBA               SUBTRACT EXPONENTS
019A 2A 02     148:         BPL   QUOT
019C 8B 0F     149:         ADD A #15         MODULO-15
019E 97 06     150: QUOT    STA A TB1PTR+1    NEW EXPONENT>>POINTER
01A0 DE 05     151:         LDX   TB1PTR      GET POINTER
01A2 A6 00     152:         LDA A X           QUOTIENT FIELD ELEMENT
01A4 39        153:         RTS
               154: ********************************************
               155: * BEGIN SEARCHING FOR ROOTS                *
               156: ********************************************
01A5 4F        157: CONT    CLR A             CLEAR "I"
01A6 97 0F     158: TEST    STA A I           SAVE CURRENT "I"
01A8 48        159:         ASL A             EXPONENT*2
01A9 81 0F     160:         CMP A #15         MODULO-15
01AB 2D 02     161:         BLT   SQ2
01AD 80 0F     162:         SUB A #15
01AF 97 06     163: SQ2     STA A TB1PTR+1
01B1 DE 05     164:         LDX   TB1PTR
01B3 A6 00     165:         LDA A X           GET SQUARED FIELD ELEMENT
01B5 97 0E     166:         STA A TEMP1       AND SAVE IT
01B7 D6 0D     167:         LDA B S1EXP       GET EXPONENT OF S1
01B9 5B 0F     168:         ADD B I
01BB C1 0F     169:         CMP B #15         MODULO-2 ADD
01BD 2D 02     170:         BLT   MUL1
01BF C0 0F     171:         SUB B #15
01C1 D7 06     172: MUL1    STA B TB1PTR+1    SAVE EXPONENT
01C3 DE 05     173:         LDX   TB1PTR      GET POINTER
01C5 E6 00     174:         LDA B X           GET FIELD ELEMENT
01C7 D8 0E     175:         EOR B TEMP1       FIND PARTIAL SUM (GF(2**4))
01C9 D8 08     176:         EOR B SIGMA2      ADD IN SIGMA2
01CB 27 04     177:         BEQ   ROOT        ROOT FOUND ?
01CD A6 0F     178: RT1     LDA A I           FETCH LOCATOR
01CF 81 0E     179:         CMP A #14
```



```
01D1 27 03     180:         BEQ   STOP        STOP IF SEARCH DONE
01D3 4C        181:         INC A             INCREMENT LOCATOR
01D4 20 D0     182:         BRA   TEST        AND TEST AGAIN
01D6 39        183: STOP    RTS               SEARCH COMPLETE
01D7 5F        184: ROOT    CLR B             CLEAR SHIFT COUNTER
01D8 74 0000   185:         LSR   RX          RIGHT JUSTIFY R(X)
01DB 76 0001   186:         ROR   RX+1
01DE D1 0F     187: TST2    CMP B I           IS SHIFT COUNT = I?
01E0 27 0F     188:         BEQ   BTSWAP      YES; CORRECT ERRONEOUS BIT
01E2 06        189: ROOT1   TAP               GET CARRY BIT
01E3 76 0000   190:         ROR   RX          ROTATE R(X) RIGHT
01E6 76 0001   191:         ROR   RX+1
01E9 07        192:         TPA               SAVE CARRY
01EA 5C        193:         INC B             BUMP COUNTER
01EB C1 10     194:         CMP B #16         16 SHIFTS DONE ?
01ED 27 DE     195:         BEQ   RT1         YES; EXIT
01EF 20 ED     196:         BRA   TST2        LOOP AGAIN
01F1 37        197: BTSWAP  PSH B             SAVE COUNTER ON STACK
01F2 D6 01     198:         LDA B RX+1
01F4 C8 01     199:         EOR B #1          SWITCH THE BIT IN ERROR
01F6 D7 01     200:         STA B RX+1        AND RESTORE R(X)
01F8 33        201:         PUL B             RESTORE STACK
01F9 20 E7     202:         BRA   ROOT1
               203:         END
     NO ERROR(S) DETECTED
```

listings for chapter 7

(1) Transmitter software

```
                        1:         NAM   MODEM TRANSMITTER SOFTWARE
                        2:         OPT   NOS,LIS,PAG
                        3: ********************************************
                        4: *
                        5: *         HF MODEM TRANSMITTER SOFTWARE
                        6: *         =============================
                        7: *
                        8: * TRANSMISSION FORMAT:
                        9: *
                       10: * (1) NOISE ESTIMATION (NO TRANSMISSION)
                       11: * (2) SYNC BLOCK (PHASE REVERSALS (179 ELS.)
                       12: *     M-SEQUENCE (31 ELS.)
                       13: *     ADVISORY SEQUENCE (30 ELS.) (FIXED FREQ.)
                       14: * (3) 53 MESSAGE BLOCKS (14880 ELS.)
                       15: *
                       16: *  BLOCK FORMAT:
                       17: * 240 ELEMENTS, 16 DATA SUBCHANNELS,
                       18: * 2 DATA SUBCHANNELS PER SUBCHANNEL FREQUENCY
                       19: * (DIFFERENTIAL 4-PHASE SHIFT KEYING).
                       20: *
                       21: ********************************************
                       22: *
F000                   23: BASE   EQU   $F000        EPROM OFFSET
C000                   24: LATCH  EQU   $C000        SLAVE CONTROL LATCH
                       25: *
0FF8                   26:        ORG   $0FF8
FFF8 FB F0             27:        FDB   IRQRTN+BASE IRQ VECTOR
0FFE                   28:        ORG   $0FFE
FFFE F9 AD             29:        FDB   START+BASE  MASTER RESET VECTOR
                       30: *
                       31: * I/O ADDRESSES:
                       32: *
6C00                   33: INPIA  EQU   $6C00        MODULATOR INPUT PIA
6C04                   34: OUTPIA EQU   $6C04        MODULATOR OUTPUT PIA
3014                   35: ADC    EQU   $3014        RECEIVER AUDIO
3010                   36: TXRX   EQU   $3010        TX/RX RELAY
300C                   37: ACIA   EQU   $300C        SERIAL INPUT TO MODEM
300D                   38: ACIAC  EQU   $300D        ACIA CONTROL REGISTER
                       39: *
                       40: * MASTER PROCESSOR VARIABLE STORAGE
                       41: *
0000                   42:        ORG   0
0000                   43: TBL1   RMB   32          CHANNEL SPECTRUM TABLE
0020                   44: TBL2   RMB   3           3 OPTIMUM SUBCHANNEL SLOTS
0023                   45: XTEMP  RMB   2           INDEX REG TEMP STORES
0025                   46: XTEMP1 RMB   2
0027                   47: XTEMP2 RMB   2
0029                   48: FROM   RMB   2
002B                   49: TO     RMB   2
002D                   50: LKINDX RMB   2           LOOKUP TABLE POINTER
002F                   51: BINDX  RMB   2           DATA POINTERS FOR FFT
0031                   52: CINDX  RMB   2
0033                   53: REAL   RMB   2           REAL FOURIER COEFFICIENT
0035                   54: IMAG   RMB   2           IMAG FOURIER COEFFICIENT
0037                   55: Y      RMB   2           MULTIPLY PARAMETERS
0039                   56: Z      RMB   2
003B                   57: P      RMB   4
003F                   58: TB     RMB   1
0040                   59: I      RMB   1           INDICES
0041                   60: M      RMB   1
0042                   61: L      RMB   1
0043                   62: MIN    RMB   2
0045                   63: COUNT1 RMB   1
0046                   64: SHFCNT RMB   1           SHIFT COUNT
0047                   65: CONTR  RMB   2           SUBCHANNEL COUNTER
0049                   66: PHPTR  RMB   2           PHASE POINTER
004B                   67: ELCNT  RMB   1           SIGNAL ELEMENT COUNTER
004C                   68: BITPTR RMB   1           BIT POINTER
004D                   69: BYTPTR RMB   1           BYTE POINTER
004E                   70: DPTR   RMB   2           DATA POINTER
0050                   71: TABPT  RMB   1
0051                   72: BUFPTR RMB   2           BUFFER POINTER
0053                   73: DIBIT  RMB   1           DIBIT
0054                   74: DTFLAG RMB   1           DATA TABLE FLAG
0055                   75: ADVSEQ RMB   16          ADVISORY SEQUENCE
0065                   76: DTBL1  RMB   30
0083                   77: DTBL2  RMB   30
0100                   78:        ORG   $0100
0100                   79: DBASE  RMB   256
0200                   80: CHN000 RMB   30          DATA SUBCHANNEL TABLES (1)
021E                   81: CHN001 RMB   30
023C                   82: CHN002 RMB   30
025A                   83: CHN003 RMB   30
0278                   84: CHN004 RMB   30
0296                   85: CHN005 RMB   30
02B4                   86: CHN006 RMB   30
02D2                   87: CHN007 RMB   30
02F0                   88: CHN008 RMB   30
030E                   89: CHN009 RMB   30
032C                   90: CHN010 RMB   30
034A                   91: CHN011 RMB   30
0368                   92: CHN012 RMB   30
0386                   93: CHN013 RMB   30
03A4                   94: CHN014 RMB   30
03C2                   95: CHN015 RMB   30
03E0                   96: CHN100 RMB   30          DATA SUBCHANNEL TABLES (2)
03FE                   97: CHN101 RMB   30
041C                   98: CHN102 RMB   30
043A                   99: CHN103 RMB   30
0458                  100: CHN104 RMB   30
0476                  101: CHN105 RMB   30
0494                  102: CHN106 RMB   30
04B2                  103: CHN107 RMB   30
04D0                  104: CHN108 RMB   30
04EE                  105: CHN109 RMB   30
050C                  106: CHN110 RMB   30
0529                  107: CHN111 RMB   30
0548                  108: CHN112 RMB   30
0566                  109: CHN113 RMB   30
0584                  110: CHN114 RMB   30
05A2                  111: CHN115 RMB   30
05C0                  112: BUFF   RMB   512         INPUT DATA BUFFER
                      113: *
                      114: * SLAVE VARIABLE STORAGE
                      115: *
0001                  116:        ORG   $0001
0001                  117: FLAG   RMB   1           MASTER HANDSHAKE
0002                  118: LU     RMB   2           LOOKUP POINTER
0004                  119: XTEMP3 RMB   2           XREG TEMP STORE
```

```
0006            120: STEP    RMB   1           LOOKUP STEP LENGTH
0007            121: SLOPG   EQU   *           SLAVE PROGRAM ORIGIN
0007            122: LKUP1   RMB   80          SPACE FOR LOOKUPS
0057            123: CHNTBL  RMB   16
0067            124: $BASE   RMB   160
                125: ************************************************
                126: * EPROM CONTENTS:
                127: ************************************************
0300            128:         ORG   $0300
0300 01 00      129: LKUP    FDB   $0100,$00FD,$00FA,$00F4,$00EB,$00E0,$00D4,$00C5
0310 00 B4      130:         FDB   $00B4,$00A1,$008D,$0078,$0061,$004A,$0031,$0018
0320 00 00      131:         FDB   $0000,$FFE6,$FFCD,$FFB4,$FF9D,$FF86,$FF71,$FF5D
0330 FF 4A      132:         FDB   $FF4A,$FF39,$FF2A,$FF1E,$FF13,$FF0A,$FF04,$FF01
0340 00 00      133:         FDB   $0000,$FFE6,$FFCD,$FFB4,$FF9D,$FF86,$FF71,$FF5D
0350 FF 4A      134:         FDB   $FF4A,$FF39,$FF2A,$FF1E,$FF13,$FF0A,$FF04,$FF01
0360 FF 00      135:         FDB   $FF00,$FF01,$FF04,$FF0A,$FF13,$FF1E,$FF2A,$FF39
0370 FF 4A      136:         FDB   $FF4A,$FF5D,$FF71,$FF86,$FF9D,$FFB4,$FFCD,$FFE6
0380 E8         137: GENPOL  FCB   %11101000,%10000000 GENERATOR POLYNOMIAL
0381            138: SLVLKP  EQU   *
0381 7F 00      139:         FDB   $7F00,$7D00,$7300,$7100,$6600,$5900,$4A00,$3900
0391 27 00      140:         FDB   $2700,$1300,$0000,$EBFF,$D7FF,$C5FF,$B4FF,$A5FF
0341 99 FF      141:         FDB   $99FF,$8DFF,$86FF,$81FF,$80FF,$81FF,$86FF,$8DFF
0381 99 FF      142:         FDB   $99FF,$A5FF,$B4FF,$C5FF,$D7FF,$EBFF,$0000,$1300
0301 27 00      143:         FDB   $2700,$3900,$4A00,$5900,$6600,$7100,$7300,$7D00
                144: *
                145: * SLAVE ROUTINES:
                146: *
03D2            147: SLVRST  EQU   *
03D2 0F         148:         SEI
03D3 8E 03E0    149:         LDS   #$03E0        SLAVE STACK POINTER
03D6 CE FF07    150:         LDX   #$FF07        INITIALISE I/O
03D9 FF 6000    151:         STX   INPIA         CA1 IS I/P (CAUSES IRQ)
03DC CE FF2C    152:         LDX   #$FF2C
03DF FF 6002    153:         STX   INPIA+2       CB2 IS O/P ("SHIFT")
03E2 CE 0004    154:         LDX   #$0004
03E5 FF 6004    155:         STX   OUTPIA
03E8 CE F004    156:         LDX   #$F004        4 MSB'S ARE CONTROL LINES
03EB FF 6006    157:         STX   OUTPIA+2
03EE 7F 6006    158:         CLR   OUTPIA+2      STOP SAMPLING CLOCK
03F1 86 30      159:         LDA A #%00110000
03F3 B7 6006    160:         STA A OUTPIA+2      GO
03F6 CE 0007    161:         LDX   #LKUP1        INITIALISE LOOKUP POINTER
03F9 DF 02      162:         STX   LU
03FB CE 0057    163: SLAVE0  LDX   #CHNTBL
03FE DF 29      164: SLAVE2  STX   XTEMP1        CURRENT CHANNEL POINTER
0400 96 00      165:         LDA A X             GET STEP LENGTH
0402 97 06      166:         STA A STEP          AND SAVE IT
0404 A6 03      167:         LDA A 3,X           GET START POINT IN LOOKUP
0406 97 03      168:         STA A LU+1          AND SAVE IN POINTER
0408 C6 28      169:         LDA B #40           ACCB IS SHIFT COUNTER
040A DE 02      170: SLAVE1  LDX   LU            GET LOOKUP POINTER
040C A6 00      171:         LDA A X             GET LOOKUP
040E BB 6004    172:         ADD A OUTPIA        LSBYTE
0411 B7 6000    173:         STA A INPIA         OTHER END OF SHIFT REG
0414 A6 01      174:         LDA A 1,X           MSBYTE (FF OR 00)
0416 B9 6006    175:         ADC A OUTPIA+2
0419 B7 6002    176:         STA A INPIA+2       SAVE AND SHIFT TO NEXT
041C 96 06      177:         LDA A STEP          GET CURRENT STEP LENGTH
041E 9B 03      178:         ADD A LU+1          ADD TO POINTER LSB
0420 81 57      179:         CMP A #87           MODULO-40 (*2,COS 2 BYTES/POINT)
0422 2D 02      180:         BLT   NOSUB1        IF <80, DON'T SUBTRACT
0424 80 50      181:         SUB A #80           IF >=80, SUBTRACT 80
0426 97 03      182: NOSUB1  STA A LU+1          NEW POINTER VALUE
0428 5A         183:         DEC B               DEC SHIFT COUNT
0429 26 DF      184:         BNE   SLAVE1        IF 40 NOT DONE, DO AGAIN
042B C6 28      185:         LDA B #40           RESTORE SHIFT REG TO START
042D 4F         186:         CLR A               BY SHIFTING 40 TIMES
042E B7 6C02    187: SLAVE6  STA A INPIA+2
0431 5A         188:         DEC B
0432 26 FA      189:         BNE   SLAVE6
0434 DE 29      190:         LDX   XTEMP1        GET CHANNEL POINTER
0436 08         191:         INX                 AND BUMP IT
0437 8C 005E    192:         CPX   #CHNTBL+7     DONE 7 CHANNELS ?
043A 26 C2      193:         BNE   SLAVE2        NO: DO NEXT CHANNEL
043C A6 00      194:         LDA A X             GET LAST STEP LENGTH
043E 97 06      195:         STA A STEP
0440 A6 03      196:         LDA A 3,X           GET INITIAL POINT IN LOOKUP
0442 97 03      197:         STA A LU+1          SAVE LSB
0444 CE 0067    198:         LDX   #$BASE        RAM TABLE BASE
0447 DF 29      199:         STX   XTEMP1        SAVE IN POINTER
0449 7A 0001    200:         DEC   FLAG          REQUEST MORE DATA
044C C6 28      201:         LDA B #40           READY FOR LAST TIME ROUND
044E 20 04      202:         BRA   SLAVE3        HOP OVER ISLAND
0450 0E         203: SLAVE5  CLI
0451 3E         204:         WAI                 WAIT FOR INTERRUPT
0452 20 A7      205:         BRA   SLAVE0
0454 37         206: SLAVE3  PSH B               SAVE COUNTER ON STACK
0455 DE 02      207:         LDX   LU            LOOKUP POINTER
0457 A6 00      208:         LDA A X             GET LSB
0459 E6 01      209:         LDA B 1,X           GET MSB
045B BB 6C04    210:         ADD A OUTPIA        ADD IN LAST VALUE
045E F9 6C06    211:         ADC B OUTPIA+2
0461 CB 04      212:         ADD B #$04          CONVERT TO STRAIGHT BINARY
0463 B7 6C00    213:         STA A INPIA         SHOVE IN OTHER END
0466 F7 6C02    214:         STA B INPIA+2       AND SHIFT
0469 DE 29      215:         LDX   XTEMP1        RAM TABLE POINTER
046B A7 00      216:         STA A X
046D E7 01      217:         STA B 1,X
046F 08         218:         INX                 BUMP TABLE POINTER
0470 08         219:         INX
0471 DF 29      220:         STX   XTEMP1        NEW VALUE
0473 96 06      221:         LDA A STEP          GET STEP
0475 9B 03      222:         ADD A LU+1          AND COMPUTE NEW
0477 81 57      223:         CMP A #87           POINTER VALUE MOD-40
0479 2D 02      224:         BLT   NOSUB2        AS BEFOR
047B 80 50      225:         SUB A #80
047D 97 03      226: NOSUB2  STA A LU+1
047F 33         227:         PUL B               GET COUNTER OFF STACK
0480 5A         228:         DEC B               DONE 40 ?
0481 26 D1      229:         BNE   SLAVE3        IF NOT, DO NEXT
0483 CE 0067    230:         LDX   #$BASE
0486 C6 28      231:         LDA B #40
0488 A6 00      232: SLAVE4  LDA A X
048A B7 6C00    233:         STA A INPIA
048D A6 01      234:         LDA A 1,X
048F B7 6C02    235:         STA A INPIA+2
0492 08         236:         INX
0493 08         237:         INX
0494 5A         238:         DEC B
0495 26 F1      239:         BNE   SLAVE4
```

```
0997 20 B7      240:          BRA   SLAVE5
                241: *
                242: * SLAVE PROCESSOR INTERRUPT ROUTINE:
                243: *
0999            244: SLIRQ    EQU   *
0999 F6 6C06    245:          LDA B OUTPIA+2     GET CONTROL LINES
099C C8 10      246:          EOR B #%00010000   SWAP SHIFT REGISTERS
099E F7 6C06    247:          STA B OUTPIA+2     AND RESTORE LINES
09A1 F6 6C00    248:          LDA B INPIA        CLEAR IRQ FLAG
09A4 3B         249:          RTI
                250: *
                251: * SLAVE VECTORS
                252: *
09A5 C0 C7      253: SLVEC    FDB   SLIRQ-SLVRST+LATCH
09A7 00 07      254:          FDB   SLORG
09A9 00 07      255:          FDB   SLORG
09AB 00 07      256:          FDB   SLORG
                257: ************************************************
                258: *  MAIN PROCEDURE BEGINS:
                259: ************************************************
09AD CE 0004    260: START    LDX   #$0004       INITIALISE I/O
09B0 FF 3014    261:          STX   ADC
09B3 CE FF2C    262:          LDX   #$FF2C
09B6 FF 3016    263:          STX   ADC+2
09B9 CE FF04    264:          LDX   #$FF04       TX/RX RELAY CONTROL
09BC FF 3010    265:          STX   TXRX
09BF 86 FF      266:          LDA A #$FF         TXMIT
09C1 B7 3010    267:          STA A TXRX
09C4 86 80      268:          LDA A #$80
09C6 B7 300D    269:          STA A ACIAC
09C9 8E 1000    270:          LDS   #$1000       SYSTEM STACK
09CC 86 FE      271:          LDA A #$FE         ENSURE SLAVE IS HALTED
09CE B7 C000    272:          STA A LATCH
09D1 7F C001    273:          CLR   FLAG+LATCH
09D4 BD FB49    274:          JSR   BOOT+BASE    BOOTSTRAP LOADER
09D7 CE 05C0    275:          LDX   #BUFF
09DA DF 56      276:          STX   BUFPTR       INITIALISE BUFFER POINTER
09DC 0E         277:          CLI
09DD BD FA47    278: NMES     JSR   SFRAME+BASE  BUILD START FRAME
09E0 DE 56      279: WDATA    LDX   BUFPTR
09E2 8C 07BF    280:          CPX   #BUFF+511    BUFFER FULL ?
09E5 26 F9      281:          BNE   WDATA        NO: WAIT TIL FULL
09E7 CE 05C0    282:          LDX   #BUFF        RESTORE BUFFER POINTER
09EA DF 56      283:          STX   BUFPTR
09EC DF 2E      284:          STX   FROM
09EE CE 03E0    285:          LDX   #CHN100      SELECT TABLE
09F1 DF 30      286:          STX   TO
09F3 BD FB91    287:          JSR   INTLVE+BASE  AND INTERLEAVE DATA
09F6 BD FC1E    288:          JSR   RCVE+BASE    ASSESS NOISE IN CHANNEL
09F9 BD FC2E    289:          JSR   ADV+BASE     CONSTRUCT ADVISORY SEQUENCE
09FC 4F         290:          CLR A
09FD 97 55      291:          STA A TABPT        POINT TO TABLE 0
09FF 97 52      292:          STA A BYTPTR       BYTE 0
0A01 4C         293:          INC A
0A02 97 51      294:          STA A BITPTR       BIT 1
0A04 BD FD03    295:          JSR   CFREQ+BASE   INITIALISE FREQUENCIES
0A07 7F 0059    296:          CLR   ELCNT        CLEAR ELEMENT COUNTER
0A0A 86 FF      297:          LDA A #$FF         START TRANSMISSION
0A0C B7 C000    298:          STA A LATCH
0A0F BD FA3C    299: TX1      JSR   PHASE+BASE
0A12 7D 0050    300:          TST   ELCNT        ALL ELEMENTS TRANSMITTED ?
0A15 26 F8      301:          BNE   TX1          NO: LOOP AGAIN
0A17 BD FCE7    302:          JSR   NFREQ+BASE   JUMP TO NEW FREQUENCIES
0A1A C6 1F      303:          LDA B #31          CODEWORD FRAME COUNTER
0A1C 37         304: NXFRAM   PSH B
0A1D CE 0200    305:          LDX   #CHN000      BACK TO TABLE 0
0A20 8D 13      306:          BSR   TX0
0A22 CE 03E0    307:          LDX   #CHN100      NOW TABLE 1
0A25 8D 0E      308:          BSR   TX0
0A27 33         309:          PUL B              MORE FRAMES ?
0A28 5A         310:          DEC B
0A29 26 F1      311:          BNE   NXFRAM
0A2B BD FA8C    312: TX5      JSR   PHASE+BASE
0A2E 7D 0050    313:          TST   ELCNT        ALL ELEMENTS TRANSMITTED ?
0A31 26 F8      314:          BNE   TX5
0A33 20 A8      315:          BRA   NMES         START NEW MESSAGE
                316: *
                317: * INTERLEAVE NEXT BLOCK. THEN WAIT FOR PREVIOUS
                318: * BLOCK TO BE TRANSMITTED.
                319: *
0A35 DF 30      320: TX0      STX   TO
0A37 CE 05C0    321:          LDX   #BUFF
0A3A DF 2E      322:          STX   FROM
0A3C BD FB91    323:          JSR   INTLVE+BASE  INTERLEAVE A BLOCK
0A3F BD FA8C    324: TX2      JSR   PHASE+BASE
0A42 7D 0050    325:          TST   ELCNT
0A45 26 F8      326:          BNE   TX2
                327: ************************************************
                328: * CONSTRUCT "START" FRAME:
                329: *  PHASE REVERSAL SEQUENCE (179 ELEMENTS).
                330: *  SYNCHRONISATION SEQUENCE (31 ELEMENTS).
                331: *  FREQUENCY ADVISORY SEQUENCE (30 ELEMENTS).
                332: ************************************************
0A47 CE 0200    333: SFRAME   LDX   #CHN000
0A4A C6 10      334:          LDA B #16          DATA SUBCHANNEL COUNTER
0A4C 37         335: SFRAM0   PSH B
0A4D DF 28      336:          STX   XTEMP
0A4F 8D 12      337:          BSR   SFRAM1
0A51 96 28      338:          LDA A XTEMP
0A53 D6 29      339:          LDA B XTEMP+1
0A55 CB 1E      340:          ADD B #30          NEXT DATA SUBCHANNEL
0A57 89 00      341:          ADC A #0
0A59 97 28      342:          STA A XTEMP
0A5B D7 29      343:          STA B XTEMP+1
0A5D DE 28      344:          LDX   XTEMP
0A5F 33         345:          PUL B
0A60 5A         346:          DEC B
0A61 26 E9      347:          BNE   SFRAM0
                348: *
0A63 DE 28      349: SFRAM1   LDX   XTEMP
0A65 C6 16      350:          LDA B #22
0A67 86 FF      351:          LDA A #$FF
0A69 A7 00      352: SFRAM2   STA A X            FILL WITH 1'S
0A6B 08         353:          INX
0A6C 5A         354:          DEC B
0A6D 26 FA      355:          BNE   SFRAM2
0A6F DF 2A      356:          STX   XTEMP1       SAVE POINTER
0A71 C6 08      357:          LDA B #8
0A73 CE FA88    358:          LDX   #MSEQ+BASE
0A76 A6 00      359: SFRAM3   LDA A X
```

```
0978 05          350:          INX
0979 DF 2D       351:          STX   XTEMP2
097B DE 29       352:          LDX   XTEMP1
097D A7 00       353:          STA A X
097F 08          354:          INX
0980 DF 29       355:          STX   XTEMP1
0982 DE 2C       356:          LDX   XTEMP2
0984 5A          357:          DEC B
0985 26 EF       358:          BNE   SFRAM3
0987 39          359:          RTS
0988 E9          360: MSEQ     FCB   $E9,$C5,$E5,$18
                 361: ********************************************
                 362: * FIND DIFFERENTIAL PHASES FROM DATA TABLE.
                 363: * LOAD ABSOLUTE PHASES INTO SLAVE.
                 364: ********************************************
098C 7D C001     365: PHASE    TST   FLAG+LATCH  SLAVE FINISHED ?
098F 26 01       366:          BNE   PHAS11
0991 39          367:          RTS
0992 C6 08       368: PHAS11   LDA B £8
0994 D7 4C       369:          STA B CONTR       SUBCHANNEL COUNTER
0996 CE C058     370:          LDX   £CHNTBL+LATCH+1 SLAVE TABLE
0999 DF 4E       371:          STX   PHPTR
099B 7D 0055     372:          TST   TABPT       TABLE INDICATOR
099E 27 05       373:          BEQ   PHASE0
09A0 CE 03E0     374:          LDX   £CHN100     TABLE 1 TO BE DONE
09A3 20 03       375:          BRA   PHASE1
09A5 CE 0200     376: PHASE0   LDX   £CHN000     TABLE 0
09A8 DF 28       377: PHASE1   STX   XTEMP
09AA D6 52       378: PHASE2   LDA B BYTPTR      GET BYTE POINTER
09AC 4F          379:          CLR A
09AD DB 29       380:          ADD B XTEMP+1     LOCATE BYTE
09AF 99 28       381:          ADC A XTEMP
09B1 D7 29       382:          STA B XTEMP+1
09B3 97 28       383:          STA A XTEMP
09B5 96 51       384: PHAS20   LDA A BITPTR      LOCATE BIT
09B7 DE 28       385:          LDX   XTEMP
09B9 7F 0058     386:          CLR   DIBIT
09BC E6 00       387:          LDA B X           GET BYTE
09BE 59          388: PHASE3   ROL B             GET BIT
09BF 4A          389:          DEC A
09C0 26 FC       390:          BNE   PHASE3
09C2 79 0058     391:          ROL   DIBIT       INTO DIBIT
09C5 E6 1E       392:          LDA B 30,X        GET NEXT BIT
09C7 96 51       393: PHASE4   LDA A BITPTR
09C9 59          394:          ROL B
09CA 4A          395:          DEC A
09CB 26 FA       396:          BNE   PHASE4
09CD 79 0058     397:          ROL   DIBIT
09D0 DE 4E       398:          LDX   PHPTR
09D2 96 58       399:          LDA A DIBIT
09D4 27 30       400:          BEQ   PHSE9
09D6 81 01       401:          CMP A £1          DIBIT=1 ?
09D8 26 0E       402:          BNE   PHASE6      NO: TEST AGAIN
09DA 96 00       403:          LDA A X           ADVANCE PHASE BY PI/2.
09DC 8B 14       404:          ADD A £20
09DE 81 51       405:          CMP A £81
09E0 23 02       406:          BLS   PHASE5
09E2 80 50       407:          SUB A £80         MOD-80 ADDITION
09E4 A7 00       408: PHASE5   STA A X
09E6 20 1E       409:          BRA   PHSE9
```



```
09E8 81 03       420: PHASE6   CMP A £3          DIBIT=3 ?
09EA 26 0E       421:          BNE   PHASE8
09EC A6 00       422:          LDA A X           ADVANCE PHASE BY PI.
09EE 8B 28       423:          ADD A £40
09F0 81 51       424:          CMP A £81
09F2 23 02       425:          BLS   PHASE7
09F4 80 50       426:          SUB A £80
09F6 A7 00       427: PHASE7   STA A X
09F8 20 0C       428:          BRA   PHSE9
09FA A6 00       429: PHASE8   LDA A X
09FC 8B 3C       430:          ADD A £60         RETARD PHASE BY PI/2.
09FE 81 51       431:          CMP A £81
0B00 23 02       432:          BLS   PHSE8
0B02 80 50       433:          SUB A £80
0B04 A7 00       434: PHSE8    STA A X
0B06 08          435: PHSE9    INX
0B07 08          436:          INX
0B08 DF 4E       437:          STX   PHPTR
0B0A 96 28       438:          LDA A XTEMP
0B0C D6 29       439:          LDA B XTEMP+1     MOVE TO NEXT
0B0E CB 3C       440:          ADD B £60         SUBCHANNEL
0B10 89 00       441:          ADC A £0
0B12 D7 29       442:          STA B XTEMP+1
0B14 97 28       443:          STA A XTEMP
0B16 7A 004C     444:          DEC   CONTR
0B19 26 99       445:          BNE   PHAS20
0B1B 96 51       446:          LDA A BITPTR
0B1D 4C          447:          INC A
0B1E 81 09       448:          CMP A £9          8 BITS DONE ?
0B20 26 01       449:          BNE   PHSE7
0B22 4F          450:          CLR A
0B23 97 51       451: PHSE7    STA A BITPTR
0B25 96 52       452:          LDA A BYTPTR
0B27 4C          453:          INC A
0B28 81 1E       454:          CMP A £30
0B2A 26 03       455:          BNE   PHSE90
0B2C 7F 0052     456:          CLR   BYTPTR
0B2F 96 50       457: PHSE90   LDA A ELCNT        SIGNAL ELEMENT COUNTER
0B31 4C          458:          INC A
0B32 81 F0       459:          CMP A £240
0B34 27 01       460:          BEQ   PHSE88      WHOLE BLOCK DONE ?
0B36 39          461:          RTS               NO: FINISH HERE
0B37 7D 0055     462: PHSE88   TST   TABPT
0B3A 27 05       463:          BEQ   PHSE85
0B3C 7F 0055     464:          CLR   TABPT
0B3F 20 04       465:          BRA   PHSE90
0B41 86 FF       466: PHSE85   LDA A £$FF
0B43 97 55       467:          STA A TABPT
0B45 7F 0050     468: PHSE90   CLR   ELCNT
0B48 39          469:          RTS
                 470: ********************************************
                 471: * BOOTSTRAP LOADER FOR SLAVE ROUTINES:
                 472: ********************************************
0B49 CE F9D2     473: BOOT     LDX   £SLVRST+BASE
0B4C DF 3C       474:          STX   Y
0B4E CE F994     475:          LDX   £SLVEC-1+BASE
0B51 DF 3E       476:          STX   Z
0B53 CE C007     477:          LDX   £SLORG+LATCH
0B56 8D 1F       478:          BSR   BOOT10
0B58 CE F9A5     479:          LDX   £SLVEC+BASE
```

```
0858 DF 3C     430:          STX   Y
085D CE F940   431:          LDX   #SLVEC+7+BASE
0860 DF 3E     432:          STX   Z
0862 CE 03F8   433:          LDX   #$03F8+LATCH
0865 8D 10     434:          BSR   BOOT10
0867 CE F832   435:          LDX   #SLVLKP+BASE
086A DF 3C     436:          STX   Y
086C CE F8D1   437:          LDX   #SLVLKP+79+BASE
086F DF 3E     438:          STX   Z
0871 CE 0007   439:          LDX   #LKUP1+LATCH
0874 8D 01     440:          BSR   BOOT10
0876 39        441:          RTS
               442: ********************************************
               443: * BOOT CONTENTS OF ADDRESSES Y -> Z INTO RAM AREA
               444: * BEGINNING WITH XREG.
               445: ********************************************
0877 DF 40     446: BOOT10 STX   P
0879 7C 003F   447:          INC   Z+1
087C DE 3C     448: BOOT11 LDX   Y
087E 9C 3E     449:          CPX   Z
0880 27 0E     450:          BEQ   BOOT12
0882 A6 00     451:          LDA A X
0884 08        452:          INX
0885 DF 3C     453:          STX   Y
0887 DE 40     454:          LDX   P
0889 A7 00     455:          STA A X
088B 08        456:          INX
088C DF 40     457:          STX   P
088E 20 EC     458:          BRA   BOOT11
0890 39        459: BOOT12 RTS
               460: ********************************************
               461: * INTERLEAVE DATA
               462: ********************************************
0891 DE 2E     463: INTLVE LDX   FROM        "FROM" ADDRESS
0893 DF 2A     464:          STX   XTEMP1
0895 C6 07     465:          LDA B #7
0897 D7 4A     466: INTLV1 STA B COUNT1
0899 8D 23     467:          BSR   TNSFER    TRANSFER A BYTE
089B 8D 21     468:          BSR   TNSFER    AND ANOTHER
089D DE 2E     469:          LDX   FROM
089F DF 2A     470:          STX   XTEMP1
08A1 D6 4A     471:          LDA B COUNT1
08A3 5A        472:          DEC B
08A4 26 F1     473:          BNE   INTLV1    DO ALL INFO BITS
08A6 DE 2E     474:          LDX   FROM
08A8 08        475:          INX
08A9 DF 2A     476:          STX   XTEMP1
08AB C6 08     477:          LDA B #8
08AD D7 4A     478: INTLV2 STA B COUNT1
08AF 8D 0D     479:          BSR   TNSFER    DO CHECK BITS
08B1 8D 08     480:          BSR   TNSFER
08B3 DE 2E     481:          LDX   FROM
08B5 08        482:          INX
08B6 DF 2A     483:          STX   XTEMP1
08B8 D6 4A     484:          LDA B COUNT1
08BA 5A        485:          DEC B
08BB 26 F0     486:          BNE   INTLV2
08BD 39        487:          RTS
               488: *
08BE C6 08     489: TNSFER LDA B #8
```

```
08C0 DE 2A     490:          LDX   XTEMP1
08C2 A6 00     491:          LDA A X
08C4 81 FF     492:          CMP A #$FF      TOP OF BUFFER ?
08C6 26 05     493:          BNE   TNSFR1
08C8 CE 05C0   494:          LDX   #BUFF     RESTORE POINTER
08CB DF 2A     495:          STX   XTEMP1
08CD 68 00     496: TNSFR1 ASL   X
08CF DE 30     497:          LDX   TO        "TO" ADDRESS
08D1 69 00     498:          ROL   X
08D3 DE 2A     499:          LDX   XTEMP1
08D5 08        500:          INX
08D6 DF 2A     501:          STX   XTEMP1
08D8 5A        502:          DEC B
08D9 26 F2     503:          BNE   TNSFR1
08DB BD FA8C   504:          JSR   PHASE+BASE
08DE DE 56     505:          LDX   BUFPTR
08E0 8C 07BF   506:          CPX   #BUFF+511 BUFFER FULL ?
08E3 26 05     507:          BNE   TNSFR2
08E5 CE 05C0   508:          LDX   #BUFF
08E8 DF 56     509:          STX   BUFPTR
08EA DE 30     510: TNSFR2 LDX   TO
08EC 08        511:          INX             READY FOR NEXT BYTE
08ED DF 30     512:          STX   TO
08EF 39        513:          RTS
               514: ********************************************
               515: * GET CHARACTER FROM ACIA.
               516: * CODE IT, AND STORE CODEWORD.
               517: ********************************************
08F0 B6 800C   518: IRQRTN LDA A ACIA
08F3 8D 0F     519:          BSR   CODE
08F5 DE 56     520:          LDX   BUFPTR
08F7 A7 00     521:          STA A X         STORE CODEWORD
08F9 E7 01     522:          STA B 1,X       IN BUFFER
08FB 86 FF     523:          LDA A #$FF      BUFFER TERMINATOR
08FD A7 02     524:          STA A 2,X
08FF 08        525:          INX
0900 08        526:          INX
0901 DF 56     527:          STX   BUFPTR
0903 3B        528:          RTI
               529: ********************************************
               530: * FIND CODEWORD FROM DATA BITS
               531: ********************************************
0904 36        532: CODE   PSH A
0905 C6 08     533: CODE1  LDA B #8
0907 D7 4B     534:          STA B SHFCNT
0909 5F        535:          CLR B
090A F8 F881   536: EXOROP EOR B GENPOL+BASE+1
090D B8 F880   537:          EOR A GENPOL+BASE
0910 85 80     538: BITTST BIT A #$80
0912 26 F6     539:          BNE   EXOROP
0914 59        540:          ROL B
0915 49        541:          ROL A
0916 7A 004B   542:          DEC   SHFCNT
0919 26 F5     543:          BNE   BITTST
091B 16        544: CODE2  TAB
091C 32        545:          PUL A
091D 39        546:          RTS
               547: ********************************************
               548: * ASSESS NOISE IN CHANNEL:
               549: ********************************************
```

```
0C1E 7F 8010  600: RCVE   CLR   TXRX        TRANSMITTER OFF.
0C21 CE FFFF  601:        LDX   #$FFFF      DELAY TO SETTLE
0C24 09       602: RCVE1  DEX
0C25 26 FD    603:        BNE   RCVE1
0C27 BD FD04  604:        JSR   TRANS+BASE  DO 8 TRANSFORMS
0C2A BD FD18  605:        JSR   FIND+BASE   FIND CLEAREST SLOTS
0C2D 39       606:        RTS
              607: ********************************************
              608: * CONSTRUCT SUBCHANNEL ADVISORY SEQUENCE
              609: ********************************************
0C2E CE 0020  610: ADV    LDX   #TBL2       TABLE OF NUMBERS
0C31 DF 29    611:        STX   XTEMP1
0C33 4F       612:        CLR A
0C34 CE 0000  613:        LDX   #TBL1       NEW TABLE
0C37 DF 2C    614: ADV0   STX   XTEMP2
0C39 DE 29    615:        LDX   XTEMP1
0C3B 36       616:        PSH A             SAVE COUNT
0C3C 48       617:        ASL A
0C3D 48       618:        ASL A
0C3E 48       619:        ASL A
0C3F 48       620:        ASL A
0C40 98 00    621:        ADD A X
0C42 48       622:        ASL A             LEFT JUSTIFY INFO BITS
0C43 8D 8F    623:        BSR   CODE
0C45 08       624:        INX
0C46 DF 29    625:        STX   XTEMP1
0C48 DE 2C    626:        LDX   XTEMP2
0C4A A7 00    627:        STA A X
0C4C E7 01    628:        STA B 1,X
0C4E 08       629:        INX
0C4F 08       630:        INX
0C50 32       631:        PUL A             RESTORE COUNT
0C51 4C       632:        INC A             NEXT SUBCHANNEL
0C52 81 08    633:        CMP A #8
0C54 26 E1    634:        BNE   ADV0
0C56 8D 38    635:        BSR   ADV00       INTERLEAVE TO DEPTH 2
0C58 CE 021A  636:        LDX   #CHN000+26  ADD IN TO START FRAME
0C5B DF 29    637:        STX   XTEMP1
0C5D C6 04    638:        LDA B #4          FOUR-FOLD SPECTRAL REDUNDANCY
0C5F 37       639: ADV23  PSH B
0C60 CE 0059  640:        LDX   #ADVSEQ
0C63 DF 2C    641:        STX   XTEMP2
0C65 C6 04    642:        LDA B #4          4 DATA SUBCHANNELS
0C67 37       643: ADV22  PSH B
0C68 C6 04    644:        LDA B #4          4 BYTES PER DATA SUBCHANNEL
0C6A DE 2C    645: ADV21  LDX   XTEMP2
0C6C A6 00    646:        LDA A X
0C6E 08       647:        INX
0C6F DF 2C    648:        STX   XTEMP2
0C71 DE 2A    649:        LDX   XTEMP1
0C73 A7 00    650:        STA A X
0C75 08       651:        INX
0C76 DF 2A    652:        STX   XTEMP1
0C78 5A       653:        DEC B
0C79 26 EF    654:        BNE   ADV21
0C7B A6 29    655:        LDA A XTEMP1      MOVE TO NEXT SUBCNHANNEL
0C7D C6 23    656:        LDA B XTEMP1+1
0C7F CB 19    657:        ADD B #26
0C81 89 00    658:        ADC A #0
0C83 D7 29    659:        STA B XTEMP1+1
0C85 97 2A    660:        STA A XTEMP1
0C87 33       661:        PUL B
0C88 5A       662:        DEC B
0C89 26 DC    663:        BNE   ADV22
0C8B 33       664:        PUL B
0C8C 5A       665:        DEC B
0C8D 26 D0    666:        BNE   ADV23
0C8F 39       667:        RTS
              668: *
              669: *
0C90 C6 04    670: ADV00  LDA B #4
0C92 CE 0000  671:        LDX   #TBL1
0C95 DF 2A    672:        STX   XTEMP1
0C97 CE 005A  673:        LDX   #ADVSEQ
0C9A DF 2C    674:        STX   XTEMP2
0C9C 6F 03    675:        CLR   3,X
0C9E 6F 07    676:        CLR   7,X
0CA0 6F 0B    677:        CLR   11,X
0CA2 6F 0F    678:        CLR   15,X
0CA4 37       679: ADV10  PSH B             SAVE DATA CHANNEL COUNTER
0CA5 C6 07    680:        LDA B #7
0CA7 8D 1B    681:        BSR   ADV3
0CA9 DE 2A    682:        LDX   XTEMP1
0CAB 08       683:        INX
0CAC C6 08    684:        LDA B #8          DO CHECKBITS
0CAE 8D 14    685:        BSR   ADV3
0CB0 DE 2A    686:        LDX   XTEMP1
0CB2 08       687:        INX
0CB3 08       688:        INX
0CB4 08       689:        INX
0CB5 08       690:        INX
0CB6 DF 2A    691:        STX   XTEMP1
0CB8 DE 2C    692:        LDX   XTEMP2
0CBA 08       693:        INX
0CBB 08       694:        INX
0CBC 08       695:        INX
0CBD 08       696:        INX
0CBE DF 2C    697:        STX   XTEMP2
0CC0 33       698:        PUL B
0CC1 5A       699:        DEC B
0CC2 26 E0    700:        BNE   ADV10
              701: *
              702: * INTERLEAVE BITS:
              703: *
0CC4 DE 2A    704: ADV3   LDX   XTEMP1
0CC6 69 00    705:        ROL   X
0CC8 09       706:        INX
0CC9 09       707:        INX
0CCA DF 2A    708:        STX   XTEMP1
0CCC 8D 0E    709:        BSR   ADV4
0CCE DE 2A    710:        LDX   XTEMP1
0CD0 69 00    711:        ROL   X
0CD2 09       712:        DEX
0CD3 09       713:        DEX
0CD4 DF 2A    714:        STX   XTEMP1
0CD6 8D 04    715:        BSR   ADV4
0CD8 5A       716:        DEC B
0CD9 26 E9    717:        BNE   ADV3
0CDB 39       718:        RTS
              719: *
```

```
0CDC DE 2C     720: ADV4   LDX   XTEMP2
0CDE 69 03     721:       ROL   3,X
0CE0 69 02     722:       ROL   2,X
0CE2 69 01     723:       ROL   1,X
0CE4 69 00     724:       ROL   0,X
0CE6 39        725:       RTS
               726: ********************************************
               727: * SET SUBCHANNELS TO NEW FREQUENCIES
               728: ********************************************
0CE7 CE C057   729: NFREQ  LDX   ECHNTBL+LATCH
0CEA DF 2C     730:       STX   XTEMP2
0CEC CE 0020   731:       LDX   ETBL2
0CEF DF 2A     732:       STX   XTEMP1
0CF1 DE 2A     733: NFREQ1 LDX   XTEMP1
0CF3 A6 00     734:       LDA A X
0CF5 48        735:       ASL A
0CF6 3B 06     736:       ADD A E6
0CF8 08        737:       INX
0CF9 DF 2A     738:       STX   XTEMP1
0CFB DE 2C     739:       LDX   XTEMP2
0CFD A7 00     740:       STA A X
0CFF 08        741:       INX
0D00 DF 2C     742:       STX   XTEMP2
0D02 8C C05F   743:       CPX   ECHNTBL+LATCH+8
0D05 26 EA     744:       BNE   NFREQ1
0D07 39        745:       RTS
               746: ********************************************
               747: * INITIALISE SUBCHANNEL FREQUENCIES TO
               748: * ALTERNATE SLOTS:
               749: ********************************************
0D08 CE C057   750: CFREQ  LDX   ECHNTBL+LATCH
0D0B A6 06     751:       LDA A E6
0D0D A7 00     752: CFREQ1 STA A X
0D0F 3B 04     753:       ADD A E4
0D11 08        754:       INX
0D12 8C C05F   755:       CPX   ECHNTBL+LATCH+8
0D15 26 F6     756:       BNE   CFREQ1
0D17 39        757:       RTS
               758: ********************************************
               759: * ROUTINE TO SELECT 8 SMALLEST 2-BYTE NUMBERS FROM
               760: * SET OF 16
               761: ********************************************
0D18 CE 0020   762: FIND   LDX   ETBL2
0D1B DF 28     763: FIND1  STX   XTEMP
0D1D 8D 1A     764:       BSR   TEST          FIND INDEX OF SMALLEST
0D1F DE 3E     765:       LDX   Z             REMAINING NUMBER.
0D21 86 FF     766:       LDA A E$FF          REPLACE MSB WITH FF
0D23 A7 00     767:       STA A X             SO WON'T GET IT AGAIN
0D25 96 3F     768:       LDA A Z+1           DIVIDE INDEX BY 2
0D27 84 1F     769:       AND A E%00011111    DON'T WANT BITS 5-7
0D29 47        770:       ASR A
0D2A DE 28     771:       LDX   XTEMP         & STORE IN TABLE OF
0D2C A7 00     772:       STA A X             SUBCHANNEL NOS.
0D2E 8C 0027   773:       CPX   ETBL2+7       ALL DONE ?
0D31 27 03     774:       BEQ   FIND2         YES: SORT INTO ASCENDING ORDER
0D33 08        775:       INX
0D34 20 E5     776:       BRA   FIND1         NO: GET NEXT NO.
0D36 8D 26     777: FIND2  BSR   CHSORT       SORT INTO ASCENDING ORDER
0D38 39        778:       RTS
               779: ********************************************
```

```
               780: * FIND THE SMALLEST REMAINING 2-BYTE NUMBER:
               781: ********************************************
0D39 86 FF     782: TEST   LDA A E$FF          GIVE 'MIN,MIN+1' A
0D3B 97 48     783:       STA A MIN           HIGH VALUE AT FIRST
0D3D C6 10     784:       LDA B E16           NUMBER COUNTER
0D3F CE 001E   785:       LDX   ETBL1+30      TABLE OF NUMBERS
0D42 37        786: TEST4  PSH B               SAVE ON STACK
0D43 E6 01     787: TEST1  LDA B 1,X           FETCH LSB
0D45 A6 00     788:       LDA A X             & MSB
0D47 91 48     789:       CMP A MIN           MSB <= MIN ?
0D49 22 0C     790:       BHI   TEST3         NO: IGNORE
0D4B 26 04     791: TEST2  BNE   TEST5         IF NOT =, MUST BE <
0D4D D1 49     792:       CMP B MIN+1         IF MSB=MIN, TEST LSB
0D4F 22 06     793:       BHI   TEST3         IF LSB>MIN+1, IGNORE
0D51 97 48     794: TEST5  STA A MIN           MIN=MSB
0D53 D7 49     795:       STA B MIN+1         MIN+1=LSB
0D55 DF 3E     796:       STX   Z             SAVE POSITION IN Z
0D57 09        797: TEST3  DEX                 GOTO NEXT NUM
0D58 09        798:       DEX
0D59 33        799:       PUL B
0D5A 5A        800:       DEC B               ALL DONE ?
0D5B 26 E5     801:       BNE   TEST4         NO: TEST NEXT NUM
0D5D 39        802:       RTS
               803: ********************************************
               804: * 8 BYTES FROM ETBL2->ETBL2+7 ARE SORTED INTO
               805: * ASCENDING ORDER.
               806: ********************************************
0D5E CE 0020   807: CHSORT LDX   ETBL2
0D61 DF 28     808: SORT1  STX   XTEMP
0D63 E6 00     809:       LDA B X             GET A BYTE
0D65 08        810: SORT2  INX
0D66 A6 00     811:       LDA A X             GET ANOTHER
0D68 11        812:       CBA                 A>B ?
0D69 22 0B     813:       BHI   NOSWAP        YES: DON'T SWAP 'EM
0D6B E7 00     814:       STA B X             YES: SWAP 'EM ROUND
0D6D DF 3C     815:       STX   Y             SAVE X
0D6F DE 28     816:       LDX   XTEMP
0D71 A7 00     817:       STA A X
0D73 16        818:       TAB                 NEW VALUE FOR ACCB
0D74 DE 3C     819:       LDX   Y             RESTORE XREG
0D76 8C 0027   820: NOSWAP CPX   ETBL2+7       ALL DONE ?
0D79 26 EA     821:       BNE   SORT2         NO: CONTINUE
0D7B DE 28     822:       LDX   XTEMP
0D7D 08        823:       INX                 MOVE ALONG THERE
0D7E 8C 0027   824:       CPX   ETBL2+7       ALL SORTED OUT ?
0D81 26 DE     825:       BNE   SORT1         NO, GO AGAIN
0D83 39        826:       RTS
               827: ********************************************
               828: * DO 8 TRANSFORMS, SUMMING POWER SPECTRA
               829: ********************************************
0D84 8D 0C     830: TRANS  BSR   CLRTBL        CLEAR TABLE
0D86 C6 08     831:       LDA B E8
0D88 37        832: LP1    PSH B
0D89 8D 44     833:       BSR   PSPEC         GET A POWER SPECTRUM
0D8B 8D 14     834:       BSR   CHSUM         SUM POINTS
0D8D 33        835:       PUL B
0D8E 5A        836:       DEC B
0D8F 26 F7     837:       BNE   LP1
0D91 39        838:       RTS
               839: ********************************************
```

```
                    340: * CLEAR THE TABLE
                    341: ****************************************
0092 86 10          342: CLRTBL LDA A #16
0094 CE 0000        343:        LDX   #TBL1
0097 6F 00          344: LP2    CLR   X
0099 6F 01          345:        CLR   1,X
009B 08             346:        INX
009C 08             347:        INX
009D 4A             348:        DEC A
009E 26 F7          349:        BNE   LP2
00A0 39             350:        RTS
                    351: ****************************************
                    352: * SUM THE 16 POINTS FROM 450 HZ. TO 2700 HZ.
                    353: ****************************************
00A1 CE 010C        354: CHSUM  LDX   #DBASE+12
00A4 DF 3C          355:        STX   Y             USE Y AS TEMP STORE
00A6 CE 0000        356:        LDX   #TBL1
00A9 DF 3E          357:        STX   Z             USE Z AS TEMP STORE
00AB C6 10          358:        LDA B #16
00AD 37             359: CHSUM1 PSH B               SAVE POINT COUNTER
00AE DE 3C          360:        LDX   Y
00B0 C6 04          361:        LDA B #4
00B2 DB 3D          362:        ADD B Y+1           POINT TO NEW ADDRESS
00B4 D7 3D          363:        STA B Y+1
00B6 A6 00          364:        LDA A X             GET SPECTRAL DATA
00B8 E6 01          365:        LDA B 1,X
00BA DE 3E          366:        LDX   Z
00BC EB 01          367:        ADD B 1,X
00BE A9 00          368:        ADC A X             ADD IN NEW VALUES
00C0 A7 00          369:        STA A X
00C2 E7 01          370:        STA B 1,X
00C4 C6 02          371:        LDA B #2            POINT TO NEW ADDR
00C6 DB 3F          372:        ADD B Z+1
00C8 D7 3F          373:        STA B Z+1
00CA 33             374:        PUL B
00CB 5A             375:        DEC B
00CC 26 DF          376:        BNE   CHSUM1
00CE 39             377:        RTS
                    378: ****************************************
                    379: * FIND POWER SPECTRUM OF INPUT SIGNAL
                    380: ****************************************
00CF BD FEEF        381: PSPEC  JSR   SMPL+BASE     GET DATA
00D2 8D 38          382:        BSR   FFT           PERFORM TRANSFORM
00D4 CE 0100        383:        LDX   #DBASE
00D7 DF 38          384:        STX   XTEMP         POINTER
00D9 C6 20          385:        LDA B #32           COUNTER
00DB 37             386: PSPEC1 PSH B
00DC DE 38          387:        LDX   XTEMP
00DE EE 00          388:        LDX   0,X           GET REAL COMPONENT
00E0 DF 3C          389:        STX   Y
00E2 DF 3E          390:        STX   Z
00E4 BD FF85        391:        JSR   MULT16+BASE FIND R*R
00E7 96 40          392:        LDA A P
00E9 D6 41          393:        LDA B P+1
00EB 37             394:        PSH B               SAVE PARTIAL RESULT ON STACK
00EC 36             395:        PSH A
00ED DE 38          396:        LDX   XTEMP
00EF EE 02          397:        LDX   2,X           GET IMAG COMPONENT
00F1 DF 3C          398:        STX   Y
00F3 DF 3E          399:        STX   Z

00F5 BD FF85        900:        JSR   MULT16+BASE FIND I*I
00F8 32             901:        PUL A               FETCH R*R
00F9 33             902:        PUL B
00FA DB 41          903:        ADD B P+1           R*R+I*I
00FC 99 40          904:        ADC A P
00FE DE 38          905:        LDX   XTEMP
0100 A7 00          906:        STA A X             RESTORE TO TABLE
0102 E7 01          907:        STA B 1,X
0104 D6 39          908:        LDA B XTEMP+1
0106 CB 04          909:        ADD B #4            BUMP POINTER
0108 D7 39          910:        STA B XTEMP+1
010A 33             911:        PUL B               FETCH COUNTER
010B 5A             912:        DEC B
010C 26 CD          913:        BNE   PSPEC1
010E 39             914:        RTS
                    915: ********************************
                    916: * FFT KERNEL: FIXED LENGTH: 64 POINTS *
                    917: ********************************
010F CE 0100        918: FFT    LDX   #DBASE
0112 DF 34          919:        STX   BINDX         INITIALISE POINTERS
0114 DF 36          920:        STX   CINDX
0116 CE F800        921:        LDX   #LKUP+BASE
0119 DF 32          922:        STX   LKINDX
011B 4F             923:        CLR A
011C 97 46          924:        STA A M             M=0
011E 97 47          925:        STA A L             L=0
0120 4C             926:        INC A
0121 97 45          927:        STA A I             I=1
0123 BD FEC8        928: BEGIN  JSR   INDX+BASE     LOCATE PAIR OF POINTS
0126 D6 46          929:        LDA B M
0128 27 0D          930:        BEQ   NOMULT        M=0?
012A BD FEBC        931:        JSR   WN+BASE       NO: FIND ADDRESS OF LOOKUPS
012D D7 33          932:        STA B LKINDX+1      SAVE LSBYTE
012F DE 36          933:        LDX   CINDX         ADDRESS OF BUTTERFLY
0131 EE 00          934:        LDX   X             (REAL PART)
0133 DF 3C          935:        STX   Y             PUT IN MULTIPLY
0135 20 07          936:        BRA   ISLE1
0137 BD FF38        937: NOMULT JSR   NOMLT+BASE    DON'T MULTIPLY
013A 20 4E          938:        BRA   TSTM          CONTINUE
013C 20 E5          939: BEGIN1 BRA   BEGIN
013E DE 32          940: ISLE1  LDX   LKINDX
0140 EE 00          941:        LDX   X             GET COSINE LOOKUP
0142 DF 3E          942:        STX   Z             SAVE IN MULTIPLICAND
0144 BD FF85        943:        JSR   MULT16+BASE FIND R1*R2
0147 DE 41          944:        LDX   P+1           GET RESULT
0149 DF 38          945:        STX   REAL          AND SAVE
014B DE 36          946:        LDX   CINDX
014D EE 02          947:        LDX   2,X           IMAG BUTTERFLY DATA
014F DF 3C          948:        STX   Y             SAVE IN MULTIPLIER
0151 BD FF85        949:        JSR   MULT16+BASE FIND I1*R2
0154 DE 41          950:        LDX   P+1           GET RESULT
0156 DF 3A          951:        STX   IMAG          AND SAVE
0158 DE 36          952:        LDX   CINDX
015A EE 02          953:        LDX   2,X           IMAG BUTTERFLY DATA
015C DF 3C          954:        STX   Y             IN MULTIPLIER
015E DE 32          955:        LDX   LKINDX
0160 EE 40          956:        LDX   64,X          GET -SINE LOOKUP
0162 DF 3E          957:        STX   Z             PUT IN MULTIPLICAND
0164 BD FF85        958:        JSR   MULT16+BASE FIND I1*I2
0167 96 38          959:        LDA A REAL
```

```
0E69 D6 39      960:        LDA B REAL+1
0E6B D0 42      961:        SUB B P+2           SUBTRACT PRODUCT FROM
0E6D 92 41      962:        SBC A P+1           CURRENT "REAL" RESULT
0E6F 97 38      963:        STA A REAL          NEW REAL VALUE
0E71 D7 39      964:        STA B REAL+1
0E73 DE 36      965:        LDX   CINDX
0E75 EE 00      966:        LDX   X             GET REAL DATA
0E77 DF 30      967:        STX   Y             PUT IN MULTIPLIER
0E79 BD FF85    968:        JSR   MULT16+BASE FIND R1*I2
0E7C 96 3A      969:        LDA A IMAG
0E7E D6 3B      970:        LDA B IMAG+1
0E80 DB 42      971:        ADD B P+2           ADD TO CURRENT IMAG
0E82 99 41      972:        ADC A P+1
0E84 97 3A      973:        STA A IMAG          NEW IMAG VALUE
0E86 D7 3B      974:        STA B IMAG+1
0E88 8D 55      975:        BSR   NWDAT0        FIND NEW POINTS
                976: ********************************
                977: * IS M=2^(I-1)-1 ?
                978: ********************************
0E8A 8D 56      979: TSTM   BSR   INDX5         FIND 2^(I-1)
0E8C C0 01      980:        SUB B #1            2^(I-1)-1
0E8E D1 46      981:        CMP B M             M=2^(I-1)-1 ?
0E90 27 05      982:        BEQ   TSTL
0E92 7C 0046    983:        INC   M             M=M+1
0E95 20 95      984: BEGIN2 BRA   BEGIN1
                985: ********************************
                986: * IS L=2^(R-I)-1 ?
                987: ********************************
0E97 7F 0046    988: TSTL   CLR   M             M=0
0E9A C6 01      989:        LDA B #1
0E9C 86 06      990:        LDA A #6            R=6
0E9E 90 45      991:        SUB A I             R-I
0EA0 8D 46      992:        BSR   INDX6         2^(R-I)
0EA2 C0 01      993:        SUB B #1            2^(R-I)-1
0EA4 D1 47      994:        CMP B L             =L ?
0EA6 27 05      995:        BEQ   TSTI
0EA8 7C 0047    996:        INC   L             L=L+1
0EAB 20 E8      997:        BRA   BEGIN2
                998: ********************************
                999: * I=R ?
               1000: ********************************
0EAD 7F 0047    1001: TSTI   CLR   L            L=0
0EB0 96 45      1002:        LDA A I
0EB2 81 06      1003:        CMP A #6           I=R ?
0EB4 27 05      1004:        BEQ   ENFFT        IF FINISHED:RETURN
0EB6 7C 0045    1005:        INC   I            I=I+1
0EB9 20 DA      1006:        BRA   BEGIN2
                1007: ********************************
0EBB 39         1008: ENFFT  RTS
                1009: ********************************
                1010: * FINDS 2*M*2^(R-I). RESULT IN ACCB
                1011: ********************************
0EBC 86 06      1012: WN     LDA A #6           R=6
0EBE 90 45      1013:        SUB A I            R-I
0EC0 27 04      1014:        BEQ   WN2          =0 ?
0EC2 58         1015: WN1    ASL B              FIND M*2^(R-I)
0EC3 4A         1016:        DEC A
0EC4 26 FC      1017:        BNE   WN1
0EC6 58         1018: WN2    ASL B              2*M*2^(R-I)
0EC7 39         1019:        RTS
```

```
                1020: ********************************
                1021: * FINDS CURRENT & BUTTERFLY LOCATIONS
                1022: ********************************
0EC8 96 45      1023: INDX   LDA A I
0ECA D6 47      1024:        LDA B L
0ECC 58         1025: INDX1  ASL B              FIND L*2^I
0ECD 4A         1026:        DEC A
0ECE 26 FC      1027:        BNE   INDX1
0ED0 DB 46      1028:        ADD B M            FIND M+L*2^I
0ED2 58         1029:        ASL B
0ED3 58         1030:        ASL B              MULT. BY 4
0ED4 D7 35      1031:        STA B BINDX+1      CURRENT INDEX
0ED6 8D 0A      1032:        BSR   INDX5        FIND 2^(I-1)
0ED8 58         1033:        ASL B              MULT BY 4
0ED9 58         1034:        ASL B
0EDA DB 35      1035:        ADD B BINDX+1      FIND M+L*2^I+2^(I-1)
0EDC D7 37      1036:        STA B CINDX+1
0EDE 39         1037:        RTS
                1038: ********************************
0EDF 8D 63      1039: NWDAT0 BSR   NWDATA
0EE1 39         1040:        RTS
                1041: * FINDS 2^(I-1)
                1042: ********************************
0EE2 C6 01      1043: INDX5  LDA B #1
0EE4 96 45      1044:        LDA A I
0EE6 80 01      1045:        SUB A #1           I-1
0EE8 27 04      1046: INDX6  BEQ   INDX3        IF 0 RETURN 1
0EEA 58         1047: INDX2  ASL B              FIND 2^(I-1)
0EEB 4A         1048:        DEC A
0EEC 26 FC      1049:        BNE   INDX2
0EEE 39         1050: INDX3  RTS
                1051: ****************************************
                1052: * SUBROUTINE TO FETCH 64 SAMPLES AT SAMPLE FREQ OF
                1053: * FC/168. SAMPLES ARE RECEIVED IN 2'S COMP AND ARE
                1054: * SCALED BY FACTOR LOG2(N) (=6) AND SAVED IN BIT-
                1055: * REVERSED ORDER IN DATA TABLE. IMAG COMPONENTS IN
                1056: * DATA TABLE ARE CLEARED.
                1057: ****************************************
0EEF CE 0100    1058: SMPL   LDX   #DBASE       SAVE ADDRESS OF
0EF2 DF 28      1059:        STX   XTEMP        TABLE BASE
0EF4 86 03      1060:        LDA A #3           SELECT MUX CHANNEL
0EF6 B7 8016    1061:        STA A ADC+2        INITIATE 1ST CONVERSION
0EF9 C6 12      1062:        LDA B #18          1ST TIMING PARAMETER
0EFB 01         1063: SMPL2  NOP                WAIT FOR CONVERSION
0EFC 5A         1064:        DEC B
0EFD 26 FC      1065:        BNE   SMPL2
0EFF 5F         1066:        CLR B              ACCB IS SAMPLE COUNTER (0->63)
0F00 17         1067: SMPL1  TBA                TRANSFER TO ACCA
0F01 37         1068:        PSH B              & SAVE ON STACK
0F02 C6 02      1069: SHUF   LDA B #%00000010   SET BIT 1 OF ACCB
0F04 44         1070: SHUF1  LSR A              TRANSFER BITS TO ACCB IN
0F05 59         1071:        ROL B              REVERSE ORDER UNTIL B7 SET
0F06 2A FC      1072:        BPL   SHUF1        WHEN DONE, B6=0, B7=1
0F08 58         1073:        ASL B              SHIFT OUT B7 & B6
0F09 58         1074:        ASL B              & MULT BY 4
0F0A D7 29      1075:        STA B XTEMP+1      ACCB IS LSB OF ADDR
0F0C DE 28      1076:        LDX   XTEMP        SET ADDRESS
0F0E B6 8014    1077:        LDA A ADC          GET DATA FROM A-D
0F11 C6 03      1078:        LDA B #3           MUX CHANNEL SELECT
0F13 F7 8016    1079:        STA B ADC+2        INITIATE CONVERSION
```

```
0F16 5F       1030:         CLR B               ACCB IS MSB OF SCALED DATA
0F17 48       1031:         ASL A               ROTATE DATA LEFT
0F18 59       1032:         ROL B               MSBIT TO ACCB
0F19 48       1033:         ASL A
0F1A 59       1034:         ROL B
0F1B A7 01    1035:         STA A 1,X           SAVE LSB OF SCALED DATA
0F1D C5 02    1036:         BIT B #%00000010    -VE ?
0F1F 26 02    1037:         BNE   NEG           YES: BRANCH
0F21 20 03    1038:         BRA   SCL1          NO: DO NOTHING
0F23 CB FC    1039: NEG     EOR B #%11111100    SET REMAINING BITS
0F25 01       1040:         NOP                 WAIT EXTRA TIME
0F26 E7 00    1041: SCL1    STA B ,X            NOW STORE MSB
0F28 6F 02    1042:         CLR   2,X           CLEAR IMAG COMPONENT
0F2A 6F 03    1043:         CLR   3,X
0F2C C6 07    1044:         LDA B #7            TIMING LOOP
0F2E 5A       1045: SMPL3   DEC B               WAIT FOR FULL PERIOD
0F2F 26 FD    1046:         BNE   SMPL3         MAKE UP TO 186 CYCLES
0F31 33       1047:         PUL B               GET COUNTER OFF STACK
0F32 5C       1048:         INC B               AND BUMP IT
0F33 C1 40    1049:         CMP B #64           FINISHED ?
0F35 26 C9    1050:         BNE   SMPL1         NO: GET NEXT SAMPLE
0F37 39       1051:         RTS                 YES: RETURN: BUFFER FULL.
              1052: ********************************
              1053: * DON'T MULTIPLY BY LOOKUPS
              1054: ********************************
0F38 DE 36    1055: NOMLT   LDX   SINDX         BUTTERFLY
0F3A EE 00    1056:         LDX   ,X            DATA GOES DIRECTLY
0F3C DF 38    1057:         STX   REAL          INTO "REAL"
0F3E DE 36    1058:         LDX   SINDX         AND "IMAG"
0F40 EE 02    1059:         LDX   2,X           WITHOUT NEED FOR
0F42 DF 3A    1060:         STX   IMAG          MULTIPLYING
              1061: ********************************
              1062: * FIND 2 NEW POINTS FROM 2 OLD
              1063: ********************************
0F44 DE 34    1064: NWDATA  LDX   SINDX         GET CURRENT DATA PTR
0F46 A6 00    1065:         LDA A ,X            REAL COMPONENT
0F48 E6 01    1066:         LDA B 1,X
0F4A 97 3C    1067:         STA A Y             Y IS TEMP STORE
0F4C D7 3D    1068:         STA B Y+1
0F4E DB 39    1069:         ADD B REAL+1        ADD TO REAL PRODUCT
0F50 99 38    1070:         ADC A REAL
0F52 A7 00    1071:         STA A ,X            SAVE IN CURRENT POSITION
0F54 E7 01    1072:         STA B 1,X
0F56 A6 02    1073:         LDA A 2,X           GET IMAG DATA
0F58 E6 03    1074:         LDA B 3,X
0F5A 97 3E    1075:         STA A Z             Z IS TEMP STORE
0F5C D7 3F    1076:         STA B Z+1
0F5E DB 3B    1077:         ADD B IMAG+1        ADD TO IMAG PRODUCT
0F60 99 3A    1078:         ADC A IMAG
0F62 DE 34    1079:         LDX   SINDX
0F64 A7 02    1080:         STA A 2,X           SAVE IN CURRENT POSITION
0F66 E7 03    1081:         STA B 3,X
0F68 96 3C    1082:         LDA A Y             GET OLD DATA (REAL)
0F6A D6 3D    1083:         LDA B Y+1
0F6C D0 39    1084:         SUB B REAL+1        SUBTRACT REAL PRODUCT
0F6E 92 38    1085:         SBC A REAL
0F70 DE 36    1086:         LDX   SINDX         BUTTERFLY DATA POINTER
0F72 A7 00    1087:         STA A ,X            SAVE REAL RESULT
0F74 E7 01    1088:         STA B 1,X
0F76 96 3E    1089:         LDA A Z             GET OLD DATA (IMAG)
```

```
0F78 D6 3F    1140:         LDA B Z+1
0F7A D0 3B    1141:         SUB B IMAG+1        SUBTRACT IMAG PRODUCT
0F7C 92 3A    1142:         SBC A IMAG
0F7E DE 36    1143:         LDX   SINDX         BUTTERFLY
0F80 A7 02    1144:         STA A 2,X           SAVE IMAG RESULT
0F82 E7 03    1145:         STA B 3,X
0F84 39       1146:         RTS
              1147: ************************************
              1148: * 16 BY 16 MULTIPLICATION ROUTINE USING BOOTH'S
              1149: * ALGORITHM.
              1150: *  MULTIPLIER: Y,Y+1
              1151: *  MULTIPLICAND: Z,Z+1
              1152: *  PRODUCT: P,P+1,P+2,P+3,P+4
              1153: *   MULTIPLIER DESTROYED
              1154: *   MULTIPLICAND PRESERVED
              1155: ************************************
0F85 CE 0000  1156: MULT16  LDX   #0            CLEAR REGS.
0F88 DF 40    1157:         STX   P
0F8A DF 42    1158:         STX   P+2
0F8C 7F 0044  1159:         CLR   TB            CLEAR TEST BYTE
0F8F CE 0010  1160:         LDX   #16
0F92 96 3D    1161: LOOP    LDA A Y+1
0F94 84 01    1162:         AND A #1
0F96 16       1163:         TAB
0F97 98 44    1164:         EOR A TB            Y(LSB)=Y(LSB-1) ?
0F99 27 1D    1165:         BEQ   SHIFT         YES: SHIFT
0F9B 5D       1166:         TST B
0F9C 27 0E    1167:         BEQ   ADD           Y(LSB)=0
0F9E 96 40    1168: SUBTR   LDA A P
0FA0 D6 41    1169:         LDA B P+1
0FA2 D0 3F    1170:         SUB B Z+1
0FA4 92 3E    1171:         SBC A Z
0FA6 97 40    1172:         STA A P
0FA8 D7 41    1173:         STA B P+1
0FAA 20 0C    1174:         BRA   SHIFT
0FAC 96 40    1175: ADD     LDA A P
0FAE D6 41    1176:         LDA B P+1
0FB0 DB 3F    1177:         ADD B Z+1
0FB2 99 3E    1178:         ADC A Z
0FB4 97 40    1179:         STA A P
0FB6 D7 41    1180:         STA B P+1
0FB8 7F 0044  1181: SHIFT   CLR   TB            CLEAR TEST BYTE
0FBB 76 003C  1182:         ROR   Y
0FBE 76 003D  1183:         ROR   Y+1
0FC1 79 0044  1184:         ROL   TB
0FC4 77 0040  1185:         ASR   P
0FC7 76 0041  1186:         ROR   P+1
0FCA 76 0042  1187:         ROR   P+2
0FCD 76 0043  1188:         ROR   P+3
0FD0 09       1189:         DEX                 DEC SHIFT COUNT
0FD1 26 BF    1190:         BNE   LOOP
0FD3 39       1191:         RTS
              1192:         END
    NO ERROR(S) DETECTED
```

listings for chapter 8

(1) Transmitter software

(2) Receiver software

```
                    1:         NAM   TRANSMITTER SOFTWARE
                    2:         OPT   NOG,PAG,LIS
                    3: ********************************************************
                    4: *
                    5: *  TRANSMITTER SOFTWARE - MODIFIED FOR SINGLE
                    6: *  SUBCHANNEL ASK TRANSMISSION.
                    7: *
                    8: ********************************************************
                    9: *
3010                10: KEYS    EQU   $3010       MORSE KEY RELAY
                    11: *
                    12: *
                    13: * VECTORS:
                    14: *
F000                15: BASE    EQU   $F000       EPROM OFFSET
C000                16: LATCH   EQU   $C000       SLAVE CONTROL LINES
BFFE                17:         ORG   $0FFE
0FFE FF F0 00       18: RESET   FDB   INITLSE+$F000 MASTER RESET
                    19: *
                    20: * I/O ADDRESSES:
6C00                21: INPIA   EQU   $6C00       SHIFT REGISTER INPUT (SLAVE)
6C04                22: OUTPIA  EQU   $6C04       SHIFT REGISTER OUTPUT (SLAVE)
                    23: *
                    24: * MASTER RAM PATCH:
                    25:         ORG   $0000
0000                26: XTEMP   RMB   2           XREG TEMP STORE
0002                27: X       RMB   1
0003                28: TEMP    RMB   2
0005                29: Y       RMB   2
0007                30: Z       RMB   2
0009                31: P       RMB   4
                    32: *
                    33: * SLAVE RAM PATCH
                    34: *
0001                35:         ORG   $0001
0001                36: FLAG    RMB   1           INDICATOR TO MASTER
0002                37: LU      RMB   2           LOOKUP POINTER
0004                38: XTEMP1  RMB   2
0006                39: STEP    RMB   1           LOOKUP STEP LENGTH
0007                40: LKUP1   RMB   30          SPACE FOR LOOKUPS
0025                41: CHNTBL  RMB   16
0035                42: IBASE   RMB   160
0100                43: SLORG   EQU   *           SLAVE PROGRAM ORIGIN
                    44: *
                    45: * PROM ORIGIN (MASTER):
                    46: *
0300                47:         ORG   $0300
0300                48: SLVLKP  EQU   *           LOOKUPS FOR SUBCARRIER GENERATION
0300 00 00          49:         FDB   $0000,$1300,$2700,$3900,$4400,$5900,$6600,$7100
0310 71 00          50:         FDB   $7300,$7D00,$7F00,$7D00,$7800,$7100,$6600,$5900
0320 44 00          51:         FDB   $4400,$3900,$2700,$1300,$0000,$EBFF,$D7FF,$C5FF
0330 B4 FF          52:         FDB   $B4FF,$A5FF,$93FF,$83FF,$36FF,$81FF,$80FF,$81FF
0340 86 FF          53:         FDB   $86FF,$83FF,$93FF,$A5FF,$B4FF,$C5FF,$D7FF,$EBFF
0350                54: SLVRST  EQU   *
0350 0F             55:         SEI
0351 8E 0380        56:         LDS   #$0380      SLAVE STACK POINTER
0354 CE FF07        57:         LDX   #IFF07      INITIALIZE I/O
0357 FF 6C01        58:         STX   INPIA       CA1 IS I/P (CAUSES IRQ)
035A CE FF3C        59:         LDX   #IFF3C
```



```
035D FF 6C02        60:         STX   INPIA+2     CB2 IS O/P ("SHIFT")
0360 CE 0004        61:         LDX   #$0004
0363 FF 6C04        62:         STX   OUTPIA
0366 CE F004        63:         LDX   #$F004      4 MSB'S ARE CONTROL LINES
0369 FF 6C06        64:         STX   OUTPIA+2
036C 7F 6C06        65:         CLR   OUTPIA+2    STOP SAMPLING CLOCK
036F 86 30          66:         LDA A #%00110000
0371 B7 6C06        67:         STA A OUTPIA+2    GO
0374 CE 0007        68:         LDX   #LKUP1      INITIALIZE LOOKUP POINTER
0377 DF 02          69:         STX   LU
0379 CE 0057        70: SLAVE0  LDX   #CHNTBL
037C DF 04          71: SLAVE2  STX   XTEMP1      CURRENT CHANNEL POINTER
037E 96 00          72:         LDA A X           GET STEP LENGTH
0380 97 06          73:         STA A STEP        AND SAVE IT
0382 A6 03          74:         LDA A 3,X         GET START POINT IN LOOKUP
0384 97 03          75:         STA A LU+1        AND SAVE IN POINTER
0386 C6 28          76:         LDA B #40         #OF BITS IN SHIFT COUNTER
0388 DE 02          77: SLAVE1  LDX   LU          GET LOOKUP POINTER
038A A6 00          78:         LDA A X           GET LOOKUP
038C BB 6C04        79:         ADD A OUTPIA      LSB/TE
038F B7 6C00        80:         STA A INPIA       OTHER END OF SHIFT REG
0392 A6 01          81:         LDA A 1,X         MSBYTE (FF OR 00)
0394 B9 6C05        82:         ADC A OUTPIA+2
0397 B7 6C02        83:         STA A INPIA+2     SAVE AND SHIFT TO NEXT
039A 96 06          84:         LDA A STEP        GET CURRENT STEP LENGTH
039C 9B 03          85:         ADD A LU+1        ADD TO POINTER LSB
039E 81 57          86:         CMP A #87         MODULO-40 (*2,FOR 2 BYTES/POINT)
03A0 2D 02          87:         BLT   NOSUB1      IF <80, DON'T SUBTRACT
03A2 80 50          88:         SUB A #80         IF >=80, SUBTRACT 80
03A4 97 03          89: NOSUB1  STA A LU+1        NEW POINTER VALUE
03A6 5A             90:         DEC B             DEC SHIFT COUNT
03A7 26 DF          91:         BNE   SLAVE1      IF 40 NOT DONE, DO AGAIN
03A9 C6 28          92:         LDA B #40         RESTORE SHIFT REG TO START
03AB 4F             93:         CLR A             BY SHIFTING 40 TIMES
03AC B7 6C02        94: SLAVE6  STA A INPIA+2
03AF 5A             95:         DEC B
03B0 26 FA          96:         BNE   SLAVE6
03B2 DE 04          97:         LDX   XTEMP1      GET CHANNEL POINTER
03B4 08             98:         INX               AND BUMP IT
03B5 8C 005A        99:         CPX   #CHNTBL+3   DONE 3 CHANNELS ?
03B8 26 C2         100:         BNE   SLAVE2      NO; DO NEXT CHANNEL
03BA A6 00         101:         LDA A X           GET LAST STEP LENGTH
03BC 97 06         102:         STA A STEP
03BE A6 03         103:         LDA A 3,X         GET INITIAL POINT IN LOOKUP
03C0 97 03         104:         STA A LU+1        SAVE LSB
03C2 CE 0057       105:         LDX   #IBASE      RAM TABLE BASE
03C5 DF 04         106:         STX   XTEMP1      SAVE IN POINTER
03C7 7A 0001       107:         DEC   FLAG
03CA C6 28         108:         LDA B #40         READY FOR LAST TIME ROUND
03CC 20 04         109:         BRA   SLAVE3      HOP OVER ISLAND
03CE 0E            110: SLAVE5  CLI
03CF 3E            111:         WAI               WAIT FOR INTERRUPT
03D0 20 A7         112:         BRA   SLAVE0
03D2 37            113: SLAVE3  PSH B             SAVE COUNTER ON STACK
03D3 DE 02         114:         LDX   LU          LOOKUP POINTER
03D5 A6 00         115:         LDA A X           GET LSB
03D7 E6 01         116:         LDA B 1,X         GET MSB
03D9 BB 6C04       117:         ADD A OUTPIA      ADD IN LAST VALUE
03DC F9 6C05       118:         ADC B OUTPIA+2
03DF CB 04         119:         ADD B #$04        CONVERT TO STRAIGHT BINARY
```

```
03E1 B7 6000  120:        STA A INPIA       SHOVE IN OTHER END
03E4 F7 6002  121:        STA B INPIA+2     AND SHIFT
03E7 DE 04    122:        LDX   XTEMP1      RAM TABLE POINTER
03E9 A7 00    123:        STA A X
03EB E7 01    124:        STA B 1,X
03ED 08       125:        INX               BUMP TABLE POINTER
03EE 08       126:        INX
03EF DF 04    127:        STX   XTEMP1      NEW VALUE
03F1 96 06    128:        LDA A STEP        GET STEP
03F3 9B 03    129:        ADD A LU+1        AND COMPUTE NEW
03F5 81 57    130:        CMP A #87         POINTER VALUE MOD-40
03F7 2D 02    131:        BLT   NOSUB2      AS BEFOR
03F9 80 50    132:        SUB A #80
03FB 97 03    133: NOSUB2 STA A LU+1
03FD 33       134:        PUL B             GET COUNTER OFF STACK
03FE 5A       135:        DEC B             DONE 40 ?
03FF 26 D1    136:        BNE   SLAVE3      IF NOT, DO NEXT
0401 CE 0067  137:        LDX   #SBASE
0404 C6 80    138:        LDA B #40
0406 A6 00    139: SLAVE4 LDA A X
0408 B7 6000  140:        STA A INPIA
040B A6 01    141:        LDA A 1,X
040D B7 6002  142:        STA A INPIA+2
0410 08       143:        INX
0411 08       144:        INX
0412 5A       145:        DEC B
0413 26 F1    146:        BNE   SLAVE4
0415 39 B7    147:        BRA   SLAVE5
              148: *
              149: * SLAVE PROCESSOR INTERRUPT ROUTINE:
              150: *
0417          151: SLIRQ  EQU   *
0417 F6 6006  152:        LDA B OUTPIA+2    GET CONTROL LINES
041A C8 10    153:        EOR B #%00010000  SWAP SHIFT REGISTERS
041C F7 6006  154:        STA B OUTPIA+2    AND RESTORE LINES
041F F6 6000  155:        LDA B INPIA       CLEAR IRQ FLAG
0422 3B       156:        RTI
              157: *
              158: * SLAVE VECTORS
              159: *
0423 01 CE    160: SLVEC  FDB   SLIRQ-SLVRST+SLORG
0425 01 07    161:        FDB   SLVRST-SLVRST+SLORG
0427 01 07    162:        FDB   SLVRST-SLVRST+SLORG
0429 01 07    163:        FDB   SLVRST-SLVRST+SLORG
              164: *
              165: * BEGIN MAIN PROCEDURE
              166: *
042B CE FF04  167: INITLSE LDX  #$FF04       INITIALISE SWITCHES
042E FF 8010  168:        STX   KEYS
0431 8E 0300  169:        LDS   #$0300       MASTER STACK POINTER
0434 86 FF    170:        LDA A #$FF
0436 B7 8010  171:        STA A KEYS         TX OFF, CW KEY OFF.
0439 86 FE    172:        LDA A #$FE
043B B7 C000  173:        STA A LATCH        BRING RST ON SLAVE LOW
043E BD FA69  174:        JSR   BOOT+BASE    BOOT IN SLAVE PROCEDURES
0441 86 FE    175: START  LDA A #$FE
0443 B7 C000  176:        STA A LATCH
0446 86 07    177:        LDA A #7           INITIALISE TO ZERO PHASE
0448 BD FACD  178:        JSR   CLOAD+BASE
044B 86 14    179:        LDA A #20          SET UP TONE ON 1500 HZ.
```



```
044D BD FA8F  180:        JSR   SLOT6+BASE
0450 86 FF    181:        LDA A #$FF
0452 B7 C000  182:        STA A LATCH
0455 86 FD    183:        LDA A #$FD         TX ON, CW KEY OFF
0457 B7 8010  184:        STA A KEYS
045A BD F99C  185:        JSR   MORSE+BASE   ANNOUNCE STATION I.D.
045D 7F 8010  186:        CLR   KEYS         TX ON, KEY ON
0460 86 FF    187:        LDA A #$FF
0462 B7 C000  188:        STA A LATCH
0465 CE 0800  189:        LDX   #2048
0468 BD FAD4  190: TONE   JSR   WAIT+BASE    SEND TONE
046B 09       191:        DEX
046C 26 FA    192:        BNE   TONE
              193: ********************************************
              194: * SEND 8 LOTS OF THE FOLLOWING:
              195: *  MESSAGE 1: PREAMBLE, 4 CHARACTER SEQUENCES, NO CODING.
              196: *  MESSAGE 2: PREAMBLE, 4 CHARACTER SEQUENCES, BCH
              197: *             CODING, NO INTERLEAVING.
              198: *  MESSAGE 3: PREAMBLE, 4 CHARACTER SEQUENCES, BCH
              199: *             CODING, INTERLEAVING TO DEPTH 16.
              200: ********************************************
046E C6 08    201: MSEN   LDA B #8
0470 37       202: MSEN5  PSH B              SHOVE COUNTER ON STACK
0471 BD F9CF  203:        JSR   MSEN1+BASE   DO 1ST MESSAGE
0474 BD F9CF  204:        JSR   MSEN1+BASE   DO IT AGAIN
0477 BD F9E5  205:        JSR   MSEN2+BASE   DO 2ND MESSAGE
047A BD F9E5  206:        JSR   MSEN2+BASE   DO IT AGAIN
047D BD FA0D  207:        JSR   MSEN3+BASE   DO 3RD MESSAGE
0480 BD FA0D  208:        JSR   MSEN3+BASE   DO IT AGAIN
0483 33       209:        PUL B              GET COUNTER
0484 5A       210:        DEC B              DO WHOLE LOT 8 TIMES
0485 26 E9    211:        BNE   MSEN5
0487 7E F941  212:        JMP   START+BASE   REPEAT FROM START
              213: *
              214: * MORSE CODE TRANSMISSION ROUTINE:
              215: *
048A 02       216: MMES   FCB   $2,$2A,$A1,$40,$3A,$2A,$11
0491 45       217:        FCB   $45,$14,$54,$22,$AA,$A1,$51,$55 REPRESENTATION O
F MESSAGE
0499 08       218:        FCB   $8,$AA,$80   'DE G8BLD'
049C B6 8010  219: MORSE  LDA A KEYS
049F 8A 01    220:        ORA A #%00000001   ENSURE KEY IS OFF
04A1 B7 8010  221:        STA A KEYS
04A4 CE F98A  222:        LDX   #MMES+BASE   GET START OF MORSE MESSAGE
04A7 C6 08    223: MRS0   LDA B #8           BIT COUNT
04A9 A6 00    224:        LDA A X            GET A BYTE
04AB 37       225: MRS1   PSH B
04AC 48       226:        ASL A              GET OUT BIT
04AD 24 08    227:        BCC   DLY          IF NOT SET, NO CHANGE
04AF F6 8010  228:        LDA B KEYS
04B2 C8 01    229:        EOR B #1           SWITCH MORSE KEY
04B4 F7 8010  230:        STA B KEYS
04B7 8D 0B    231: DLY    BSR   SPACE        GO TO WAIT LOOP
04B9 33       232: MRS2   PUL B
04BA 5A       233:        DEC B              GET NEXT BIT
04BB 26 EE    234:        BNE   MRS1
04BD 08       235:        INX                GET NEXT BYTE
04BE 8C F99D  236:        CPX   #MMES+BASE+13
04C1 26 E4    237:        BNE   MRS0         GO TO END OF MESSAGE
04C3 39       238:        RTS
04C4 DF 00    239: SPACE  STX   XTEMP         SAVE XREG
```

```
0916 CE 9999  240:        LDX   £9000       LOOP PARAMETER
0919 09       241: SPC1   DEX
091A 26 FD    242:        BNE   SPC1
091C DE 00    243:        LDX   XTEMP       RESTORE XREG
091E 39       244:        RTS
              245: ********************************************
              246: * MESSAGE 1: SEND PREAMBLE, 4 CHAR SEQUENCES.
              247: ********************************************
091F 8D 5D    248: MSEN1  BSR   PRBLE       SYNC PREAMBLE
0921 86 E8    249:        LDA A £%11101000  1ST MESSAGE I.D. BYTE
0923 C6 07    250:        LDA B £7
0925 8D F874  251:        JSR   SNBITS+BASE SEND AS SINGLE CHAR
0928 C6 04    252:        LDA B £4          CHARACTER SEQUENCE COUNTER
092A 37       253: MSEN10 PSH B
092B CE F831  254:        LDX   £MSGE1+BASE BASE OF 1ST MESSAGE
092E 8D 6F    255:        BSR   STRING      SEND AS A STRING
0930 33       256:        PUL B
0931 5A       257:        DEC B
0932 26 F6    258:        BNE   MSEN10
0934 39       259:        RTS
              260: ********************************************
              261: * MESSAGE 2: SEND PREAMBLE, 4 CHAR SEQUENCES CODED WITH
              262: *            NO INTERLEAVING:
              263: ********************************************
0935 8D 47    264: MSEN2  BSR   PRBLE       SYNC PREAMBLE
0937 86 A6    265:        LDA A £%10100110  2ND MESSAGE I.D.
0939 C6 07    266:        LDA B £7
093B 8D F874  267:        JSR   SNBITS+BASE SEND AS SINGLE CHAR
093E C6 04    268:        LDA B £4          SEQUENCE COUNTER
0940 37       269: MSEN20 PSH B
0941 CE F852  270:        LDX   £MSGE2+BASE
0944 A6 00    271: MSEN21 LDA A X           GET CHAR (7 BITS)
0946 81 23    272:        CMP A £'£         END OF SEQUENCE ?
0948 27 0E    273:        BEQ   MSEN22      YES: SHOVE OFF
094A C6 07    274:        LDA B £7          NO: SEND 7 BITS
094C 8D 75    275:        BSR   SNBITS
094E 08       276:        INX
094F A6 00    277:        LDA A X           GET BYTE (CHECKBITS)
0951 C6 08    278:        LDA B £8
0953 8D 6F    279:        BSR   SNBITS      AND SEND THEM
0955 08       280:        INX
0956 20 EC    281:        BRA   MSEN21
0958 33       282: MSEN22 PUL B
0959 5A       283:        DEC B
095A 26 E4    284:        BNE   MSEN20
095C 39       285:        RTS
              286: ********************************************
              287: * MESSAGE 3: PREAMBLE, 4 CHAR SEQUENCES BCH ENCODED,
              288: *            INTERLEAVED TO DEPTH 16
              289: ********************************************
095D 8D 1F    290: MSEN3  BSR   PRBLE       SEND SYNC PREAMBLE
095F 86 9C    291:        LDA A £%10011100  3RD MESSAGE I.D.
0961 C6 07    292:        LDA B £7
0963 8D F874  293:        JSR   SNBITS+BASE
0966 C6 04    294:        LDA B £4
0968 37       295: MSEN30 PSH B
0969 CE F83E  296:        LDX   £MSGE3+BASE
096C A6 00    297: MSEN31 LDA A X
096E 81 23    298:        CMP A £'£
0970 27 07    299:        BEQ   MSEN32
```



```
0972 C6 08    300:        LDA B £8
0974 8D 4E    301:        BSR   SNBITS
0976 08       302:        INX
0977 20 F3    303:        BRA   MSEN31
0979 33       304: MSEN32 PUL B
097A 5A       305:        DEC B
097B 26 EB    306:        BNE   MSEN30
097D 39       307:        RTS
              308: ********************************************
              309: * SEND PREAMBLE AND/OR STRING:
              310: ********************************************
097E C6 80    311: PRBLE  LDA B £128
0980 37       312: PRBLE1 PSH B             SEND REVERSALS
0981 8D FAD4  313:        JSR   WAIT+BASE
0984 C6 14    314:        LDA B £20
0986 8D 77    315:        BSR   SLOT6
0988 8D FAD4  316:        J R   WAIT+BASE
098B 4F       317:        CLR A
098C 8D 71    318:        BSR   SLOT6
098E 33       319:        PUL B
098F 5A       320:        DEC B
0990 26 EE    321:        BNE   PRBLE1
0992 CE FA5B  322:        LDX   £MSEQ+BASE  SEND PREAMBLE
0995 A6 00    323:        LDA A X
0997 8D 2B    324:        BSR   SNBITS
0999 08       325:        INX
099A A6 00    326:        LDA A X
099C 8D 26    327:        BSR   SNBITS
099E 39       328:        RTS
              329: ********************************************
              330: * SEND STRING OF ASCII CHARS
              331: ********************************************
099F A6 00    332: STRING LDA A X
09A1 81 23    333:        CMP A £'£
09A3 27 05    334:        BEQ   STRNG0
09A5 8D 06    335:        BSR   CHAR
09A7 08       336:        INX
09A8 20 F5    337:        BRA   STRING
09AA 39       338: STRNG0 RTS
              339: *
09AB 8F 59    340: MSEQ   FDB   $8F59
              341: ********************************************
              342: * SEND AN ASCII CHARACTER:
              343: ********************************************
09AD DF 03    344: CHAR   STX   TEMP          SAVE XREG
09AF 36       345:        PSH A               SAVE CHARACTER
09B0 8D 72    346:        BSR   WAIT
09B2 86 14    347:        LDA A £20
09B4 8D 49    348:        BSR   SLOT6         START BIT
09B6 32       349:        PUL A
09B7 48       350:        ASL A               LEFT JUSTIFY CHARACTER
09B8 C6 07    351:        LDA B £7            BIT COUNTER
09BA 8D 08    352:        BSR   SNBITS        SEND CHARACTER BITS
09BC 8D 66    353:        BSR   WAIT
09BE 4F       354:        CLR A               STOP BIT
09BF 8D 3E    355:        BSR   SLOT6
09C1 DE 03    356:        LDX   TEMP          RESTORE XREG
09C3 39       357:        RTS
              358: ********************************************
              359: * SEND THE NO. OF LEFT-JUSTIFIED BITS IN ACCA INDICATED
```

```
                    360: * BY COUNT IN ACCB
                    361: ********************************************
0474 DF 00          362: INBITS STX   XTEMP        SAVE XREG
0476 49             363: ROTAT  ROL A              GET OUT BIT
0477 36             364:        PSH A              SAVE REST
0478 37             365:        PSH B              SAVE COUNT
0479 25 03          366:        BCS   CON          CHANNELS ON IF '1'
047B 4F             367: COFF   CLR A              CHANNELS OFF IF '0'
047C 20 02          368:        BRA   NXBIT
047E 86 14          369: CON    LDA A #20
0480 8D 52          370: NXBIT  BSR   WAIT         WAIT FOR ELEMENT
0482 8D 28          371:        BSR   SLOT6
0484 33             372:        PUL B
0485 32             373:        PUL A              RESTORE STACK
0486 5A             374:        DEC B
0487 26 ED          375:        BNE   ROTAT
0489 DE 00          376:        LDX   XTEMP        RESTORE XREG
048B 39             377:        RTS
                    378: ********************************************
                    379: * SET UP STEP LENGTHS IN SLAVE RAM
                    380: ********************************************
048C CE C057        381: CSET   LDX   #CHNTBL+LATCH
048F C6 24          382:        LDA B #36          FIRST STEP
0491 D7 06          383:        STA B STEP
0493 C6 04          384:        LDA B #4           CHANNEL COUNTER
0495 37             385: CSET1  PSH B
0496 D6 06          386:        LDA B STEP
0498 46             387:        ROR A              GET DATA BIT
0499 24 06          388:        BCC   CSET2        '0' ?
049B E7 00          389:        STA B X            NO: CHANNEL NEEEDED
049D 6F 04          390:        CLR   4,X
049F 20 04          391:        BRA   CSET3
04A1 6F 00          392: CSET2  CLR   X            NOT NEEDED, SO
04A3 6F 04          393:        CLR   4,X          CLEAR SLOTS
04A5 C0 08          394: CSET3  SUB B #8           NEXT STEP LENGTH
04A7 D7 06          395:        STA B STEP
04A9 08             396:        INX
04AA 33             397:        PUL B
04AB 5A             398:        DEC B
04AC 26 E7          399:        BNE   CSET1
04AE 39             400:        RTS
                    401: ********************************************
                    402: * SET UP A SINGLE CHANNEL
                    403: ********************************************
04AF CE C057        404: SLOT6  LDX   #CHNTBL+LATCH
04B2 C6 04          405:        LDA B #4
04B4 A7 00          406: SLOT60 STA A X
04B6 6F 04          407:        CLR   4,X
04B8 08             408:        INX
04B9 5A             409:        DEC B
04BA 26 F8          410:        BNE   SLOT60
04BC 39             411:        RTS
                    412: ********************************************
                    413: * SUBR TO CONVERT BINARY PHASE INDICATOR TO
                    414: * LOOKUP START INDEX:
                    415: ********************************************
04BD 4D             416: SINLKP TST A
04BE 26 03          417:        BNE   TST1         00?
04C0 86 07          418:        LDA A #7           YES: 00->07
04C2 39             419:        RTS
```



```
04C3 81 01          420: TST1   CMP A #1
04C5 26 03          421:        BNE   TST2        01?
04C7 86 1B          422:        LDA A #27          YES: 01->27
04C9 39             423:        RTS
04CA 81 02          424: TST2   CMP A #2
04CC 26 03          425:        BNE   TST3        10?
04CE 86 43          426:        LDA A #67          YES: 10->67
04D0 39             427:        RTS
04D1 86 2F          428: TST3   LDA A #47          11->47
04D3 39             429:        RTS
                    430: ********************************************
                    431: * WAIT TIL SLAVE FINISHED TASK:
                    432: ********************************************
04D4 7D C001        433: WAIT   TST   FLAG+LATCH
04D7 27 FB          434:        BEQ   WAIT
04D9 7F C001        435:        CLR   FLAG+LATCH
04DC 39             436:        RTS
                    437: ********************************************
                    438: * LOAD ALL CHANNELS WITH SPECIFIED PHASES:
                    439: ********************************************
04DD CE C05F        440: CLOAD  LDX   #CHNTBL+LATCH+8
04E0 A7 00          441: CLOAD1 STA A X
04E2 08             442:        INX
04E3 8C C067        443:        CPX   #CHNTBL+LATCH+16
04E6 26 F8          444:        BNE   CLOAD1
04E8 39             445:        RTS
                    446: *
                    447: ********************************************
                    448: * BOOTSTRAP LOADER FOR SLAVE PROCESSOR ROUTINES:
                    449: ********************************************
04E9 CE F850        450: BOOT   LDX   #SLVBOT+BASE
04EC DF 05          451:        STX   Y            BASE OF SOURCE
04EE CE F922        452:        LDX   #SLVEC-1+BASE
04F1 DF 07          453:        STX   Z            TOP OF SOURCE
04F3 CE C107        454:        LDX   #SLORG+LATCH BASE OF DESTINATION
04F6 8D 1F          455:        BSR   BOOT10       BOOT ROUTINES IN.
04F8 CE F923        456:        LDX   #SLVEC+BASE
04FB DF 05          457:        STX   Y            BASE OF SOURCE
04FD CE F98A        458:        LDX   #SLVEC+7+BASE
0500 DF 07          459:        STX   Z            TOP OF SOURCE
0502 CE C3F8        460:        LDX   #$03F8+LATCH BASE OF DESTINATION
0505 8D 10          461:        BSR   BOOT10       BOOT VECTORS IN.
0507 CE F800        462:        LDX   #SLVLKP+BASE
050A DF 05          463:        STX   Y            BASE OF SOURCE
050C CE F84F        464:        LDX   #SLVLKP+79+BASE
050F DF 07          465:        STX   Z            TOP OF SOURCE
0511 CE C007        466:        LDX   #LKUP1+LATCH BASE OF DESTINATION
0514 8D 01          467:        BSR   BOOT10       BOOT LOOKUP TABLES IN.
0516 39             468:        RTS
                    469: ********************************************
                    470: * BOOT CONTENTS OF ADDRESSES Y->Z INTO RAM AREA
                    471: * BEGINNING WITH XREG.
                    472: ********************************************
0517 DF 09          473: BOOT10 STX   P            SAVE TARGET BASE
0519 7C 0009        474:        INC   2+1
051C DE 05          475: BOOT11 LDX   Y
051E 9C 07          476:        CPX   Z
0520 27 0E          477:        BEQ   BOOT12
0522 A6 00          478:        LDA A X            GET 1 BYTE
0524 08             479:        INX
```

```
0B25 DF 05     480:         STX   Y
0B27 DE 09     481:         LDX   P
0B29 A7 00     482:         STA A X
0B2B 08        483:         INX
0B2C DF 09     484:         STX   P
0B2E 20 EC     485:         BRA   BOOT11
0B30 39        486: BOOT12 RTS
               487: *
               488: * SEQUENCE 1:
               489: *           ASCII CHARACTER STRING ONLY
               490: *
0B31 54        491: MSGE1  FCC   'THEQUICKBROWNFOXJUMPSOVERTHELAZY£'
               492: *
               493: * SEQUENCE 2:
               494: *          LEFT-JUSTIFIED ASCII CHARS FOLLOWED BY CHECKBYTES
               495: *
0B52 A3        496: MSGE2  FCB   $A3,$34,$90,$F5,$8A,$DF,$A2,$03,$AA,$E5,$92,$24
0B5E 36        497:         FCB   $36,$4A,$96,$57,$84,$9B,$A4,$A1,$3E,$D1,$AE,$96
0B6A 9C        498:         FCB   $9C,$60,$BC,$7D,$9E,$B1,$B0,$CF,$94,$86,$AA,$E5
0B76 9A        499:         FCB   $9A,$C2,$A0,$D2,$A6,$70,$9E,$B1,$AC,$47,$8A,$DF
0B82 A4        500:         FCB   $A4,$A1,$A8,$34,$90,$F5,$8A,$DF,$98,$13,$82,$39
0B8E 24        501:         FCB   $B4,$BC,$B2,$1E
0B92 23        502:         FCC   '£ TERMINATOR
               503: *
               504: * SEQUENCE 3:
               505: *        BIT-INTERLEAVED CODEWORDS (D=16)
               506: *
0B9E FF        507: MSGE3  FCB   $FF,$FF,$00,$00,$98,$51,$45,$2B,$A8,$3E,$03,$FE
0BAA 3F        508:         FCB   $3F,$32,$68,$F3,$6B,$0D,$CC,$6E,$E1,$B6,$22,$85
0BB6 ED        509:         FCB   $ED,$15,$33,$91,$79,$E7,$FF,$FF,$00,$00,$5A,$C3
0BC2 A4        510:         FCB   $A4,$2B,$67,$58,$3E,$82,$6D,$15,$F5,$B2,$73,$30
0BCE 4C        511:         FCB   $4C,$E6,$1D,$7F,$01,$17,$C3,$73,$B3,$19,$47,$BC
0BDA 23        512:         FCC   '£ TERMINATOR
               513: *
               514: * PREAMBLE SEQUENCE:
               515: *
               516:         END
     NO ERROR(S) DETECTED
```

OOO:

```
                    1:         NAM   RECEIVER SOFTWARE
                    2:         OPT   NOS,LIS,PAG
                    3: ********************************************************
                    4: * SYNCHRONISATION, DE-INTERLEAVING, AND DECODING
                    5: * ROUTINES.
                    6: *
                    7: * ANALYSIS IS PRINTED ON TERMINAL AS FOLLOWS:
                    8: *  TIME OF DAY & DATA TYPE (UNCODED, CODED, INTERLEAVED)
                    9: *  NO. OF ERRORS IN DECODED DATA STREAM
                   10: *  BURST ERROR DISTRIBUTION
                   11: *  IF DATA IS CODED, THEN:
                   12: *   NO. OF ERRORS IN RECEIVED BIT STREAM
                   13: *   TOTAL NO. OF CORRECTED ERRORS IN RECEIVED BIT STREAM
                   14: *
                   15: ********************************************************
                   16: *
                   17: * INTERRUPT VECTOR:
                   18: *
F000               19:         ORG   $F000
F000 09 0C         20:         FDB   IRQRTC
                   21: *
                   22: * MONITOR LINKAGES:
                   23: *
E101               24: OUTCH  EQU   $E101
E07E               25: PDATA  EQU   $E07E
E0C3               26: OUT4HS EQU   $E0C3
E09E               27: OUT2H  EQU   $E09E
E07B               28: INCH   EQU   $E07B
E089               29: INHEX  EQU   $E089
                   30: *
                   31: * DISC FILE MANAGEMENT LINKAGES:
                   32: *
7730               33: OPFM   EQU   $7730      OPEN DFM
7733               34: CLFM   EQU   $7733      CLOSE DFM
7736               35: DFM    EQU   $7736
7203               36: CWARMS EQU   $7203      WARM START
7249               37: CTYPDE EQU   $7249      TYPE DISC ERROR
                   38: *
                   39: * VARIABLE STORAGE AREA:
                   40: *
0000               41:         ORG   0
0000               42: HOURS  RMB   1         REAL-TIME CLOCK PARAMETERS
0001               43: MINS   RMB   1
0002               44: SECS   RMB   1
0003               45: DAY    RMB   1
0004               46: SCOUNT RMB   2         SECONDS COUNTER
0006               47: XTEMP  RMB   2         XREG TEMPORARY STORAGE
0008               48: XTEMP1 RMB   2
000A               49: ERRCNT RMB   2         BIT ERROR COUNTER
000C               50: PREV   RMB   1         'ERROR ON PREVIOUS' FLAG
000D               51: ERRPTR RMB   2         STORED DATA BUFFER POINTER
000F               52: STPTR  RMB   2         ERROR STATISTICS POINTER
0011               53: MSGNUM RMB   1         MESSAGE IDENTITY NUMBER
0012               54: MERR   RMB   2         NO. OF ERRORS IN DECODED DATA
0014               55: XTEMP2 RMB   2
0016               56: TEMP   RMB   2         TEMP STORE
0018               57: INPAT  RMB   2         RECEIVED SYNC PATTERN
                   58: *
                   59: * I/O ADDRESSES
```



```
                   60: *
F532               61: PIA    EQU   $F532
F530               62: PIAT   EQU   $F530
                   63: ********************************************
                   64: * BEGIN MAIN PROCEDURE:
                   65: ********************************************
0100               66:         ORG   $0100
0100 CE 0004       67:         LDX   #$0004        INITIALISE PIAS
0103 FF F532       68:         STX   PIA           RECEIVED DATA PORT
0106 08            69:         INX
0107 FF F530       70:         STX   PIAT          TIMER INTERRUPT PORT
010A 0F            71:         SEI                 MASK OFF INTERRUPTS
010B BD 7780       72:         JSR   OPFM          OPEN DFM
010E CE 0A37       73:         LDX   #FCB0
0111 BD 0209       74:         JSR   OPNFIL        OPEN FILE ON DRIVE 0
0114 86 01         75:         LDA A #1
0116 B7 0831       76:         STA A FCB1+2
0119 CE 082F       77:         LDX   #FCB1
011C BD 0209       78:         JSR   OPNFIL        OPEN FILE ON DRIVE 1
011F BD 0942       79:         JSR   FETTIM        AND REQUEST TIME FROM TERMINAL
0122 0E            80:         CLI                 ENABLE 1HZ. CLOCK
0123 3D 27         81: BEGIN1 BSR   INTLSE        INITIALISE VARIABLES
0125 B1 0372       82:         CMP A IDPT1         1ST MESSAGE ?
0128 26 F9         83:         BNE   BEGIN1        NO: START AGAIN
012A BD 0363       84:         JSR   MSGE1         YES: READ IT
012D BD 019C       85:         JSR   STDATA        AND ANALYSE IT
0130 3D 1A         86: BEGIN2 BSR   INTLSE        DO SAME FOR 2ND MESSAGE
0132 B1 0373       87:         CMP A IDPT2
0135 26 F9         88:         BNE   BEGIN2
0137 BD 043E       89:         JSR   MSGE2
013A BD 019C       90:         JSR   STDATA
013D 3D 0D         91: BEGIN3 BSR   INTLSE        3RD MESSAGE SEARCH
013F B1 0374       92:         CMP A IDPT3
0142 26 F9         93:         BNE   BEGIN3
0144 BD 0546       94:         JSR   MSGE3
0147 BD 019C       95:         JSR   STDATA
014A 20 D7         96:         BRA   BEGIN1        START ALL OVER AGAIN
                   97: ********************************************
                   98: * INITIALISE VARIABLES ETC. :
                   99: ********************************************
014C 86 FE        100: INTLSE LDA A #$FE          STOP BOTH SLAVES
014E B7 C000      101:         STA A LATCH
0151 B7 D000      102:         STA A LATCH1
0154 BD 017B      103:         JSR   SLOAD        LOAD TIMER SLAVE
0157 B7 D007      104:         STA A FLAG         SET FLAG IN TIMER SLAVE
015A CE 0000      105:         LDX   #0
015D DF 0A        106:         STX   ERRCNT       ZERO THE ERROR COUNT
015F DF 12        107:         STX   MERR         ZERO MESSAGE ERROR COUNT
0161 FF C004      108:         STX   ERRCOR+LATCH ZERO NO. OF CORRECTED ERRORS
0164 CE 0BD7      109:         LDX   #ERRBUF      INITIALISE ERROR STATS BUFFER
0167 DF 0F        110:         STX   STTPTR       POINTER
0169 6F 00        111:         CLR   X
016B 6F 01        112:         CLR   1,X
016D 86 FF        113:         LDA A #$FF         'END OF TABLE' FLAG
016F A7 02        114:         STA A 2,X
0171 7F 000C      115:         CLR   PREV         NO ERRORS ON PREVIOUS
0174 BD 0721      116:         JSR   SYNC         LOOK FOR SYNC PATTERN
0177 BD 0320      117:         JSR   IDPAT        GET I.D. NUMBER
017A 39           118:         RTS
                  119: ********************************************
```

```
                    120: * LOAD SLAVE
                    121: ***********************************************
0178 CE 0191        122: SLOAD  LDX   #SLDATA      'FROM'
017B DF 06          123:        STX   XTEMP
017D CE D100        124:        LDX   #$D100       'TO'
0180 DF 02          125:        STX   XTEMP1
0182 C6 0F          126:        LDA B #11
0184 BD 0584        127:        JSR   MOVE26       TRANSFER THEM
0187 CE 0130        128:        LDX   #$0100       RESET VECTOR
018A FF D3FE        129:        STX   $D3FE
018D 39             130:        RTS
018E CE             131: SLDATA FCB   $CE,$0B,$72,$09,$26,$FD,$7F,$00,$07,$20,$F5
                    132: ***********************************************
                    133: * PRINT DATA AND WRITE TO DISC:
                    134: ***********************************************
0190 BD 0240        135: STDATA JSR   CORR         CORRELATE DATA
0193 BD 0256        136:        JSR   NEWLIN
0196 BD 0256        137:        JSR   NEWLIN
0199 CE 0666        138:        LDX   #STRNG8      PRINT STARS
019C BD E07E        139:        JSR   PDATA
019F BD 0256        140:        JSR   NEWLIN
01A2 CE 0613        141:        LDX   #STRNG1
01A5 BD E07E        142:        JSR   PDATA        PRINT TIME STRING
01A8 86 FF          143:        LDA A #$FF
01AA BD 0329        144:        JSR   WDISC1       WRITE START BYTE TO DISC
01AD BD 026F        145:        JSR   TIME         READ TIME, WRITE & PRINT
01B0 96 11          146:        LDA A MESNUM
01B2 BD 030C        147:        JSR   WDISC0       WRITE MESSAGE TYPE
01B5 96 11          148:        LDA A MESNUM       GET IT AGAIN
01B7 81 01          149:        CMP A #1           TYPE APPROPRIATE STRING
01B9 26 05          150:        BNE   GDAT1
01BB CE 0625        151:        LDX   #STRNG2
01BE 20 03          152:        BRA   GDAT4
01C0 81 02          153: GDAT1  CMP A #2
01C2 26 05          154:        BNE   GDAT2
01C4 CE 0630        155:        LDX   #STRNG3
01C7 20 00          156:        BRA   GDAT4
01C9 CE 0656        157: GDAT2  LDX   #STRNG4
01CC BD E07E        158: GDAT4  JSR   PDATA
01CF BD 0256        159:        JSR   NEWLIN
01D2 CE 067E        160:        LDX   #STRNG5
01D5 BD E07E        161:        JSR   PDATA        ERROR COUNT STRING
01D8 CE 0012        162:        LDX   #MERR
01DB BD E003        163:        JSR   OUT4HS
01DE 96 12          164:        LDA A MERR         WRITE ERROR COUNT
01E0 BD 030C        165:        JSR   WDISC0
01E3 96 13          166:        LDA A MERR+1
01E5 BD 030C        167:        JSR   WDISC0
01E8 CE 071E        168:        LDX   #CRLF
01EB BD E07E        169:        JSR   PDATA
01EE BD 0645        170:        JSR   ERRPAT       WRITE ERROR PATTERNS
01F1 CE 0692        171:        LDX   #STRNG7
01F4 BD E07E        172:        JSR   PDATA
01F7 CE 0014        173:        LDX   #ERRCNT      TRUE ERROR COUNT
01FA BD E003        174:        JSR   OUT4HS
01FD 96 14          175:        LDA A ERRCNT
01FF BD 030C        176:        JSR   WDISC0
0202 96 15          177:        LDA A ERRCNT+1
0204 BD 030C        178:        JSR   WDISC0
0207 96 11          179:        LDA A MESNUM
```



```
0209 81 01          180:        CMP A #1
020B 27 1E          181:        BEQ   GTEND        NO CORRECTIONS FOR 1ST
020D CE 071E        182:        LDX   #CRLF        NEW LINE
0210 BD E07E        183:        JSR   PDATA
0213 CE 0601        184:        LDX   #STRNG6
0216 BD E07E        185:        JSR   PDATA
0219 CE 0004        186:        LDX   #ERRCOR+LATCH NO. CORRECTED
021C BD E003        187:        JSR   OUT4HS
021F B6 0004        188:        LDA A ERRCOR+LATCH
0222 BD 030C        189:        JSR   WDISC0
0225 B6 0005        190:        LDA A ERRCOR+LATCH+1
0228 BD 030C        191:        JSR   WDISC0
022B BD 0256        192: GTEND  JSR   NEWLIN       NEW LINE
022E BD E07E        193:        JSR   PDATA
0231 CE 0906        194:        LDX   #BUFF
0234 BD 025D        195:        JSR   DATPNT       PRINT DATA
0237 BD 0256        196:        JSR   NEWLIN
023A CE 0946        197:        LDX   #BUFF+64
023D BD 025D        198:        JSR   DATPNT
0240 BD 0256        199:        JSR   NEWLIN
0243 CE 06E6        200:        LDX   #STRNG8
0246 BD E07E        201:        JSR   PDATA
0249 39             202:        RTS
                    203: ***********************************************
                    204: * NEW LINE ON PRINTER
                    205: ***********************************************
024A CE 071E        206: NEWLIN LDX   #CRLF
024D BD E07E        207:        JSR   PDATA
0250 39             208:        RTS
                    209: ***********************************************
                    210: * PRINT DATA ON TERMINAL
                    211: ***********************************************
0251 C6 40          212: DATPNT LDA B #64
0253 96 00          213: PDAT0  LDA A 0
0255 81 40          214:        CMP A #$40
0257 2E 02          215:        BGT   PDAT1
0259 86 2A          216:        LDA A #'*
025B BD E1D1        217: PDAT1  JSR   OUTCH
025E 08             218:        INX
025F 5A             219:        DEC B
0260 26 F1          220:        BNE   PDAT0
0262 39             221:        RTS
                    222: ***********************************************
                    223: * READ TIME, PRINT IT, WRITE TO DISC
                    224: ***********************************************
0263 0F             225: TIME   SEI
0264 CE 0000        226:        LDX   #HOURS
0267 BD E09E        227:        JSR   OUT2H
026A 86 3A          228:        LDA A #':
026C BD E1D1        229:        JSR   OUTCH
026F CE 0001        230:        LDX   #MINS
0272 BD E09E        231:        JSR   OUT2H
0275 86 3A          232:        LDA A #':
0277 BD E1D1        233:        JSR   OUTCH
027A CE 0002        234:        LDX   #SECS
027D BD E09E        235:        JSR   OUT2H
0280 96 00          236:        LDA A HOURS        WRITE ON DISC
0282 BD 030C        237:        JSR   WDISC0       0
0285 BD 0329        238:        JSR   WDISC1       & 1 OR 2
0288 96 01          239:        LDA A MINS
```

```
0296 BD 0300   240:         JSR   WDISC0
0299 BD 0320   241:         JSR   WDISC1
029C 96 02     242:         LDA A SECS
029E BD 0300   243:         JSR   WDISC0
02A1 BD 0329   244:         JSR   WDISC1
02A4 39        245:         RTS
               246: ********************************************
               247: * WRITE ERROR PATTERN DATA TO DISC
               248: ********************************************
02A5 CE 0B37   249: ERRPAT LDX   #ERRBUF
02A8 DF 06     250:         STX   XTEMP
02AA 86 AA     251: ERST0   LDA A #$AA          'GOOD BITS' MARKER
02AC BD 06     252:         BSR   ERST10
02AE 86 BB     253:         LDA A #$BB          'BAD BITS' MARKER
02B0 BD 02     254:         BSR   ERST10
02B2 20 F6     255:         BRA   ERST0
               256: *
02B4 DE 06     257: ERST10 LDX   XTEMP
02B6 6D 00     258:         TST   X
02B8 2B 1C     259:         BMI   ERST3         OUT IF TABLE END
02BA BD 0329   260:         JSR   WDISC1        WRITE MARKER
02BD 6D 00     261:         TST   X
02BF 27 05     262:         BEQ   ERST2
02C1 A6 00     263:         LDA A X             WRITE MSB IF NEEDED
02C3 BD 0329   264:         JSR   WDISC1
02C6 DE 06     265: ERST2   LDX   XTEMP
02C8 08        266:         INX
02C9 DF 06     267:         STX   XTEMP
02CB A6 00     268:         LDA A X             WRITE LSB
02CD BD 0329   269:         JSR   WDISC1
02D0 DE 06     270:         LDX   XTEMP
02D2 08        271:         INX
02D3 DF 06     272:         STX   XTEMP
02D5 39        273:         RTS
02D6 31        274: ERST3   INS
02D7 31        275:         INS
02D8 39        276:         RTS
               277: ********************************************
               278: * OPEN FILE FOR WRITE
               279: ********************************************
02D9 DF 09     280: OPNFIL STX   XTEMP1
02DB 86 01     281:         LDA A #1
02DD A7 00     282:         STA A X
02DF 86 02     283:         LDA A #2            FILE TYPE
02E1 A7 0C     284:         STA A 12,X          (BINARY SEQUENTIAL)
02E3 BD 7786   285:         JSR   DFM
02E6 27 09     286:         BEQ   FILOPN
02E8 BD 72A9   287:         JSR   ZTYPDE
02EB BD 7783   288:         JSR   CDFM
02EE BD 7283   289:         JSR   ZWARMS
02F1 86 02     290: FILOPN LDA A #2
02F3 DE 09     291:         LDX   XTEMP1
02F5 A7 00     292:         STA A X
02F7 39        293:         RTS
               294: ********************************************
               295: * CLOSE FILE FOR WRITE:
               296: ********************************************
02F8 86 03     297: CLSFIL LDA A #3
02FA A7 00     298:         STA A X
02FC BD 7786   299:         JSR   DFM
```



```
02FF 27 09     300:         BEQ   CLSOK
0301 BD 72A9   301:         JSR   ZTYPDE
0304 BD 7783   302:         JSR   CDFM
0307 7E 7283   303:         JMP   ZWARMS
030A 0E        304: CLSOK   CLI
030B 39        305:         RTS
               306: ********************************************
               307: * WRITE TO DRIVE 0
               308: ********************************************
030C 36        309: WDISC0 PSH A
030D 37        310:         PSH B
030E DF 14     311:         STX   XTEMP2
0310 CE 0B37   312:         LDX   #FCB0
0313 BD 7786   313:         JSR   DFM
0316 27 0B     314:         BEQ   WRITOK
0318 BD 72A9   315:         JSR   ZTYPDE
031B BD 7783   316:         JSR   CDFM
031E 7E 7283   317:         JMP   ZWARMS
0321 DE 14     318:         LDX   XTEMP2
0323 33        319: WRITOK PUL B
0324 32        320:         PUL A
0325 DE 14     321:         LDX   XTEMP2
0327 0E        322:         CLI
0328 39        323:         RTS
               324: ********************************************
               325: * WRITE TO UNITS 1 OR 2
               326: ********************************************
0329 36        327: WDISC1 PSH A
032A 37        328:         PSH B
032B DF 14     329:         STX   XTEMP2
032D CE 0B2F   330:         LDX   #FCB1
0330 BD 7786   331:         JSR   DFM           TRY TO WRITE BYTE
0333 27 28     332:         BEQ   WROK
0335 B6 0B30   333:         LDA A FCB1+1        GET ERROR STATUS
0338 81 07     334:         CMP A #7            DISC FULL ?
033A 27 09     335:         BEQ   DISFUL        YES
033C BD 72A9   336: DERR    JSR   ZTYPDE        NO; MUST BE ANOTHER ERROR
033F BD 7783   337:         JSR   CDFM
0342 BD 7283   338:         JSR   ZWARMS
0345 B6 0B31   339: DISFUL LDA A FCB1+2        ON DRIVE 2 ?
0348 81 02     340:         CMP A #2
034A 27 11     341:         BEQ   WROK          BOTH FULL, DON'T WRITE
034C CE 0B2F   342:         LDX   #FCB1
034F BD 03F3   343:         JSR   CLSFIL        CLOSE FILE ON UNIT 1
0352 B6 02     344:         LDA A #2            NOW CHANGE UNIT #
0354 B7 0B31   345:         STA A FCB1+2
0357 CE 0B2F   346:         LDX   #FCB1
035A BD 02D9   347:         JSR   OPNFIL        AND OPEN FILE ON UNIT 2
035D 0E        348: WROK    CLI
035E DE 14     349:         LDX   XTEMP2
0360 33        350:         PUL B
0361 32        351:         PUL A
0362 39        352:         RTS
               353: ********************************************
               354: * FIRST MESSAGE:
               355: ********************************************
0363 CE 0382   356: MSGE1   LDX   #MES1
0366 DF 0D     357:         STX   ERRPTR
0368 CE 0A06   358:         LDX   #BUFF
036B BD 0794   359: MSGE11 JSR   CHAR
```

```
036E 87 00    360:        STA A X
0370 BC 0085  361:        CPX   #BUFF+127
0372 27 03    362:        BEQ   MSGE12
0374 08       363:        INX
0375 20 F3    364:        BRA   MSGE11
0377 86 04    365: MSGE12 LDA A #4          TERMINATOR
0379 08       366:        INX
037A A7 00    367:        STA A X
037C 86 01    368:        LDA A #1
037E B7 11    369:        STA A MESNUM      MESSAGE IDENTIFIER
0380 39       370:        RTS
0381 54       371: MES1   FCC   'THEQUICKBROWNFOXJUMPSOVERTHELAZY'
0382 54       372:        FCC   'THEQUICKBROWNFOXJUMPSOVERTHELAZY'
0383 54       373:        FCC   'THEQUICKBROWNFOXJUMPSOVERTHELAZY'
0384 54       374:        FCC   'THEQUICKBROWNFOXJUMPSOVERTHELAZY'
              375: ********************************************
              376: * GET A CHARACTER INTO ACCA (MAIN PROCESSOR TIMING):
              377: ********************************************
0402 0F       378: CHAR0  SEI
0403 4F       379:        CLR A
0404 CE 3000  380:        LDX   #12480
0407 09       381: CHAR1  DEX
0408 27 2B    382:        BEQ   CHARC
0409 7D F532  383:        TST   PIA         LOOK FOR START BIT
040C 27 F8    384:        BEQ   CHAR1
040E 8D 26    385:        BSR   WAIT        WAIT 1/2 BIT TIME
0410 C6 07    386:        LDA B #7          BIT COUNTER
0412 37       387: CHARA  PSH B
0413 8D 21    388:        BSR   WAIT
0415 8D 1F    389:        BSR   WAIT
0417 7D F532  390:        TST   PIA         GET A BIT
041A 2A 03    391:        BPL   ZEROB
041C 08       392: ONEB   INC
041D 20 01    393:        BRA   CHARB
041F 0C       394: ZEROB  CLC
0420 49       395: CHARB  ROL A
0421 33       396:        PUL B
0422 5A       397:        DEC B
0423 26 ED    398:        BNE   CHARA
0425 8D 0F    399:        BSR   WAIT        WAIT FOR STOP BIT
0427 8D 0D    400:        BSR   WAIT
0429 CE 3000  401:        LDX   #12480
042C 09       402: STOP   DEX
042D 27 05    403:        BEQ   CHARC
042F 7D F532  404:        TST   PIA
0432 2B F8    405:        BMI   STOP
0434 0E       406: CHARC  CLI
0435 39       407:        RTS
              408: ********************************************
              409: * WAIT 1/2 BIT TIME
              410: ********************************************
0437 CE 05BA  411: WAIT   LDX   #1466
043A 09       412: WAIT1  DEX
043B 26 FD    413:        BNE   WAIT1
043D 39       414:        RTS
              415: ********************************************
              416: * SECOND MESSAGE: CODED BUT NOT INTERLEAVED
              417: ********************************************
043E CE 04BD  418: MSGE3  LDX   #MES2       POINT TO STORED DATA
0441 DF 00    419:        STX   ERRPTR
```



```
0443 CE 0A06  420:        LDX   #BUFF
0446 DF 08    421:        STX   XTEMP1        INITIALISE BUFFER POINTER
0448 86 FF    422:        LDA A #$FF
044A B7 C091  423:        STA A TBFLAG+LATCH SET SLAVE STATUS
044D 7F C003  424:        CLR   TBFLAG+LATCH
0450 7F C002  425:        CLR   INFLAG+LATCH NO INTERLEAVING REQUIRED
0453 CE C05C  426:        LDX   #TBCOD0+LATCH
0456 BD 050C  427:        JSR   MSGE20        GET 16 CODEWORDS
0459 86 FF    428:        LDA A #$FF
045B B7 C060  429:        STA A LATCH         START SLAVE
045E CE C07C  430:        LDX   #TBCOD1+LATCH
0461 BD 050C  431:        JSR   MSGE20        GET 16 CODEWORDS
0464 C6 03    432:        LDA B #3
0466 37       433: MSGE22 PSH B
0467 0F       434:        SEI
0468 BD 04FD  435:        JSR   MSGE23        CHECK STATUS ETC.
046B B7 C003  436:        STA A TBFLAG+LATCH INDICATE TABLE 1
046E B7 C000  437:        STA A LATCH
0471 CE C05C  438:        LDX   #TBCOD0+LATCH
0474 DF 06    439:        STX   XTEMP
0476 BD 0532  440:        JSR   MSGE24
0479 CE 04BD  441:        LDX   #MES2         RE-INITIALISE STORED DATA POINTER
047C DF 0D    442:        STX   ERRPTR
047E CE C05C  443:        LDX   #TBCOD0+LATCH
0481 BD 050C  444:        JSR   MSGE20        GET 16 CODEWORDS
0484 0F       445:        SEI
0485 8D 76    446:        BSR   MSGE23
0487 7F C003  447:        CLR   TBFLAG+LATCH
048A B7 C000  448:        STA A LATCH
048D CE C07C  449:        LDX   #TBCOD1+LATCH
0490 DF 06    450:        STX   XTEMP
0492 BD 0532  451:        JSR   MSGE24
0495 CE C07C  452:        LDX   #TBCOD1+LATCH
0498 8D 7E    453:        BSR   MSGE20        GET 16 CODEWORDS
049A 33       454:        PUL B
049B 5A       455:        DEC B
049C 26 C8    456:        BNE   MSGE22
049E 8D 5D    457:        BSR   MSGE23        CHECK STATUS ETC.
04A0 B7 C003  458:        STA A TBFLAG+LATCH
04A3 B7 C000  459:        STA A LATCH
04A6 CE C05C  460:        LDX   #TBCOD0+LATCH
04A9 DF 06    461:        STX   XTEMP
04AB BD 0532  462:        JSR   MSGE24
04AE BD 04FD  463:        JSR   MSGE23
04B1 CE C07C  464:        LDX   #TBCOD1+LATCH
04B4 DF 06    465:        STX   XTEMP
04B6 8D 7A    466:        BSR   MSGE24
04B8 86 02    467:        LDA A #2
04BA 97 11    468:        STA A MESNUM        MESSAGE I.D.
04BC 39       469:        RTS
04BD 54       470: MES2   FCB   $54,$34,$48,$F5,$45,$DF,$51,$03,$55,$E5
04C7 49       471:        FCB   $49,$24,$43,$4A,$48,$57,$42,$98,$52,$91
04D1 4F       472:        FCB   $4F,$B1,$57,$96,$4E,$60,$46,$7D,$4F,$81
04DB 58       473:        FCB   $53,$0F,$4A,$86,$55,$E5,$4D,$03,$50,$D2
04E5 53       474:        FCB   $53,$70,$4F,$81,$56,$47,$45,$DF,$52,$41
04EF 54       475:        FCB   $54,$34,$48,$F5,$45,$DF,$4C,$13,$41,$39
04F9 5A       476:        FCB   $5A,$BC,$59,$1E
              477: ********************************************
              478: * CHECK SLAVE STATUS, TURN OFF SLAVE, RESET STATUS
              479: ********************************************
```

```
04FD 7D C001   430: MSGE23 TST   STFLAG+LATCH
0500 26 FB     431:        BNE   MSGE23
0502 86 FE     432:        LDA A #$FE
0504 B7 C000   433:        STA A LATCH
0507 4C        434:        INC A
0508 B7 C001   435:        STA A STFLAG+LATCH
050B 39        436:        RTS
               437: ********************************************
               438: * GET 16 CODEWORDS INTO BUFFER POINTED AT
               439: * BY X REGISTER:
               440: ********************************************
050C C6 10     441: MSGE20 LDA B #16          CODEWORD COUNTER
050E 37        442: MSGE29 PSH B
050F 4F        443:        CLR A
0510 C6 07     444:        LDA B #7           INFORMATION BIT COUNT
0512 BD 13     445:        BSR   MSGE21
0514 C6 07     446:        LDA B #7
0516 BD 0B75   447:        JSR   ERROR        FIND IF ERRORS IN INFOBITS
0519 C6 08     448:        LDA B #8           CHECKBIT COUNT
051B BD 04     449:        BSR   MSGE21
051D C6 08     450:        LDA B #8
051F BD 0B75   451:        JSR   ERROR        FIND IF ERRORS IN CHECKBITS
0522 33        452:        PUL B
0523 5A        453:        DEC B
0524 26 E8     454:        BNE   MSGE29
0526 39        455:        RTS
               456: ********************************************
               457: * GET BITS INTO ACCA. (NO. OF BITS IN ACCB)
               458: * STORE ACCA AT X. BUMP X
               459: ********************************************
0527 BD 0788   460: MSGE21 JSR   BIT          GET A BIT
0529 48        461:        ASL A              INTO ACCA
052A 5A        462:        DEC B              BIT COUNT
052B 26 F9     463:        BNE   MSGE21
052D A7 00     464:        STA A X            STORE BYTE
052F 08        465:        INX
0530 39        466:        RTS
               467: ********************************************
               468: * TRANSFER BYTES BETWEEN BUFFERS
               469: ********************************************
0531 C6 10     470: MSGE24 LDA B #16
0533 DE 06     471: MSGE26 LDX   XTEMP        FIRST POINTER
0535 A6 00     472:        LDA A X            'FROM' ADDRESS
0537 08        473:        INX
0538 DF 06     474:        STX   XTEMP
053A DE 08     475:        LDX   XTEMP1       2ND POINTER
053C A7 00     476:        STA A X            'TO' ADDRESS
053E 08        477:        INX
053F DF 02     478:        STX   XTEMP1
0541 5A        479:        DEC B
0542 26 EF     480:        BNE   MSGE26
0544 39        481:        RTS
               482: ********************************************
               483: * THIRD MESSAGE: CODED AND INTERLEAVED TO DEPTH 16:
               484: ********************************************
0545 CE 0504   485: MSGE3  LDX   #MES3
0548 DF 0D     486:        STX   ERRPTR
054A CE 0406   487:        LDX   #BUFF
054D DF 08     488:        STX   XTEMP1
054F 86 FF     489:        LDA A #$FF
```



```
0552 B7 C001   540:        STA A STFLAG+LATCH SET SLAVE STATUS FLAG
0555 B7 C002   541:        STA A INFLAG+LATCH INDICATE INTERLEAVING REQUIRED
0558 7F C003   542:        CLR   TBFLAG+LATCH INDICATE TABLE 0
055B CE C020   543:        LDX   #TBINT0+LATCH
055E BD 0600   544:        JSR   MSGE30       FILL TABLE 0 WITH DATA
0561 86 FF     545:        LDA A #$FF
0563 B7 C000   546:        STA A LATCH        TELL SLAVE TO DECODE IT
0566 CE C03E   547:        LDX   #TBINT1+LATCH
0569 BD 0600   548:        JSR   MSGE30       FILL TABLE 1 WITH DATA
056C C6 03     549:        LDA B #3           GROUP COUNTER
056E 37        550: MSGE32 PSH B
056F 0F        551:        SEI
0570 BD 3B     552:        BSR   MSGE23       CHECK IF SLAVE FINISHED DC.
0572 B7 C003   553:        STA A TBFLAG+LATCH INDICATE TABLE 1 TO SLAVE
0575 B7 C000   554:        STA A LATCH        THEN RESTART SLAVE
0578 CE C050   555:        LDX   #TBCOD0+LATCH
057B DF 06     556:        STX   XTEMP
057D CE 0504   557:        LDX   #MES3
0580 DF 0D     558:        STX   ERRPTR       RE-INITIALISE POINTER
0582 BD AE     559:        BSR   MSGE24       EXTRACT DATA FROM TABLE 0
0584 CE C020   560:        LDX   #TBINT0+LATCH
0587 BD 77     561:        BSR   MSGE30       GET 30 BYTES OF DATA
0589 0F        562:        SEI
058A BD 04FD   563:        JSR   MSGE23       CHECK SLAVE STATUS DC.
058D 7F C003   564:        CLR   TBFLAG+LATCH INDICATE TABLE 0 TO SLAVE
0590 B7 C000   565:        STA A LATCH        RESTART SLAVE
0593 CE C07C   566:        LDX   #TBCOD1+LATCH
0596 DF 06     567:        STX   XTEMP
0598 BD 93     568:        BSR   MSGE24       EXTRACT DATA FROM TABLE 1
059A CE C03E   569:        LDX   #TBINT1+LATCH
059D BD 61     570:        BSR   MSGE30       GET 30 BYTES OF DATA
059F 33        571:        PUL B
05A0 5A        572:        DEC B
05A1 26 CB     573:        BNE   MSGE32
05A3 BD 04FD   574:        JSR   MSGE23       CHECK SLAVE STATUS
05A6 B7 C003   575:        STA A TBFLAG+LATCH
05A9 B7 C000   576:        STA A LATCH
05AC CE C050   577:        LDX   #TBCOD0+LATCH
05AF DF 06     578:        STX   XTEMP
05B1 BD 0532   579:        JSR   MSGE24       GET OUT DATA
05B4 BD 04FD   580:        JSR   MSGE23
05B7 CE C07C   581:        LDX   #TBCOD1+LATCH
05BA DF 06     582:        STX   XTEMP
05BC BD 0532   583:        JSR   MSGE24
05BF 86 03     584:        LDA A #3
05C1 97 11     585:        STA A MESNUM       MESSAGE I.D. NUMBER
05C3 39        586:        RTS
05C4 FF        587: MES3   FCB   $FF,$FF,$00,$00,$93,$51,$45,$2B,$A9,$3E
05CE 03        588:        FCB   $03,$FE,$3F,$32,$69,$F3,$62,$0D,$CC,$65,$E1
05D9 B6        589:        FCB   $B6,$22,$35,$ED,$15,$23,$91,$79,$E7,$FF,$FF
05E4 00        590:        FCB   $00,$00,$5A,$C3,$A4,$2B,$67,$58,$3E,$32,$6D
05EF 15        591:        FCB   $15,$F5,$B2,$7B,$30,$4C,$E6,$1D,$7F,$01,$17
05FA C3        592:        FCB   $C3,$73,$B3,$19,$47,$BC
               593: ********************************************
               594: * GET 30 BYTES OF INTERLEAVED CODEWORDS:
               595: ********************************************
0600 C6 1E     596: MSGE30 LDA B #30
0602 37        597: MSGE31 PSH B
0603 4F        598:        CLR A
0604 C6 08     599:        LDA B #8
```

```
0606 BD 0627   600:        JSR   MSGE21
0609 C6 08     601:        LDA B £8
060B BD 0375   602:        JSR   ERROR       FIND ERRORS, IF ANY
060E 33        603:        PUL B
060F 5A        604:        DEC B
0610 26 F0     605:        BNE   MSGE31
0612 39        606:        RTS
               607: ********************************************
               608: * MESSAGE AREA:
               609: ********************************************
0613 41        610: STRNG1 FCC   'ANALYSIS AT TIME '
0624 04        611:        FCB   4
0625 20        612: STRNG2 FCC   ' OF DATA WITHOUT CODING'
0642 04        613:        FCB   4
0643 20        614: STRNG3 FCC   ' OF DATA WITH BCH CODING'
0655 04        615:        FCB   4
0656 20        616: STRNG4 FCC   ' OF DATA WITH BCH CODING & INTERLEAVING'
0679 04        617:        FCB   4
067A 45        618: STRNG5 FCC   'ERROR COUNT IN DATA STREAM = '
0698 04        619:        FCB   4
0699 4E        620: STRNG7 FCC   'NO. OF ERRORS IN TOTAL BIT STREAM = '
06B7 04        621:        FCB   4
06B8 4E        622: STRNG8 FCC   'NO. OF ERRORS CORRECTED = '
06CE 04        623:        FCB   4
06CF 52        624: STRNG9 FCC   'READY....'
06D9 04        625:        FCB   4
06DA 2A        626: STRNGA FCC   '**********************************************
********'
0710 04        627:        FCB   4
0711 0D        628: CRLF   FCB   $D,$A,4     C/R, L/F STRING
               629: ********************************************
               630: * SYNCHRONISE WITH START PATTERN:
               631: ********************************************
0721 C6 08     632: SYNC   LDA B £8          MUST GET 8 BITS
0723 37        633:        PSH B
0724 8D 4B     634: SYNCA  BSR   SYNC0       : SYNC ON BIT
0726 33        635:        PUL B
0727 5A        636:        DEC B
0728 26 FA     637:        BNE   SYNCA
072A CE 0580   638:        LDX   £1408       WAIT TIL CENTRE
072D 09        639: SYNCB  DEX               OF BIT
072E 26 FD     640:        BNE   SYNCB
0730 86 FF     641:        LDA A £$FF        START SLAVE
0732 B7 0007   642:        STA A FLAG
0735 B7 0000   643:        STA A LATCH1
               644: *
               645: * GET FRAME SYNC:
               646: *
0738 C6 80     647:        LDA B £128        128 TRIES ALLOWED
073A 7F 0019   648:        CLR   SNPAT
073D 7F 0018   649:        CLR   SNPAT+1
0740 37        650: SYNCC  PSH B
0741 8D 78     651:        BSR   BIT         GET A BIT
0743 79 0019   652:        ROL   SNPAT+1
0746 79 0018   653:        ROL   SNPAT
0749 96 19     654:        LDA A SNPAT
074B C6 19     655:        LDA B SNPAT+1
074D B8 0732   656:        EOR A MSE0        CORRELATE WITH
0750 F8 0733   657:        EOR B MSE0+1      STORED SEQUENCE
0753 B7 16     658:        STA A TEMP
0755 F7 17     659:        STA B TEMP+1
```



```
0757 C6 0F     660:        LDA B £15
0759 4F        661:        CLR A
075A 76 0016   662: SYNCD  ROR   TEMP
075D 76 001E   663:        ROR   TEMP1
0760 25 01     664:        BCS   SYNCE
0762 4C        665:        INC A                 CORRELATION COUNT
0763 5A        666: SYNCE  DEC B
0764 26 F4     667:        BNE   SYNCD
0766 81 03     668:        CMP A £8
0768 2E 06     669:        BGT   SYNCF           GOT SYNC ?
076A 33        670:        PUL B                 NO; TRY AGAIN
076B 5A        671:        DEC B
076C 26 D2     672:        BNE   SYNCC
076E 20 B1     673:        BRA   SYNC            REDO BIT SYNC
0770 39        674: SYNCF  RTS
               675: *
               676: * GET BIT SYNC
               677: *
0771 7D F532   678: SYNC0  TST   PIA           '0' ?
0774 26 FB     679:        BNE   SYNC0
0776 7D F532   680: SYNC1  TST   PIA           '1' ?
0779 27 FB     681:        BEQ   SYNC1
077B CE 16B6   682: SYNC2  LDX   £5814
077E 09        683:       DEX
077F 26 FA     684:        BNE   SYNC2           WAIT.
0781 7D F532   685:        TST   PIA           SHOULD BE '0' HERE
0784 26 EB     686:        BNE   SYNC0         TRY AGAIN IF NOT
0786 CE 0062   687:        LDX   £98
0789 09        688: SYNC3  DEX
078A 26 FD     689:        BNE   SYNC3           WAIT AGAIN
078C 7D F532   690:        TST   PIA           SHOULD BE '1' HERE
078F 26 E0     691:        BNE   SYNC0         TRY AGAIN IF NOT
0791 39        692:        RTS
0792 9E B2     693: MSE0   FDB   $9EB2          SYNC PATTERN
               694: ********************************************
               695: * GET A CHARACTER (SLAVE PROCESSOR TIMING):
               696: ********************************************
0794 8D 25     697: CHAR   BSR   BIT             GET A BIT
0796 25 05     698:        BCS   STTOK
0798 BD 089B   699:        JSR   BITERR          ERROR IN START BIT
079B 20 03     700:        BRA   STRT
079D BD 08DB   701: STTOK  JSR   NBITER         NO ERROR IN START BIT
07A0 4F        702: STRT   CLR A
07A1 C6 07     703:        LDA B £7              BIT COUNT
07A3 8D 16     704: DBIT   BSR   BIT
07A5 49        705:        ROL A
07A6 5A        706:        DEC B
07A7 26 FA     707:        BNE   DBIT
07A9 C6 07     708:        LDA B £7
07AB BD 0375   709:        JSR   ERROR           CHECK FOR ERRORS
07AE 8D 0B     710: STOP0  BSR   BIT
07B0 24 05     711:        BCC   STOPOK
07B2 BD 089B   712:        JSR   BITERR
07B5 20 03     713:        BRA   STOP1
07B7 BD 08DB   714: STOPOK JSR   NBITER         NO ERROR IN STOP BIT
07B9 39        715: STOP1  RTS
               716: ********************************************
               717: * GET A BIT
               718: ********************************************
07BB 0F        719: BIT    SEI                   MASK INTERRUPT
```

```
0381 7D 0007   720:          TST   FLAG
0384 26 4C     721:          BNE   BIT1
0386 CE 039E   722:          LDX   #CRLF
0389 BD E07E   723:          JSR   PDATA
038C CE 03E3   724:          LDX   #ERRMES
038F BD E07E   725:          JSR   PDATA
0392 8E F072   726:          LDS   #$F072       RELOAD STACK POINTER
0395 96 11     727:          LDA A MESNUM
0397 81 01     728:          CMP A #1
0399 27 07     729:          BEQ   B1
039B 81 02     730:          CMP A #2
039D 27 36     731:          BEQ   B2
039F 7E 0130   732:          JMP   BEGIN3
03A2 7E 0133   733: B1       JMP   BEGIN1
03A5 7E 0130   734: B2       JMP   BEGIN2
03A8 2A        735: ERRMES   FCC   '*** BIT TIMING ERROR - SLAVE RELOADED ***'
03CA 04        736:          FCB   4
03CB 7D 0007   737: BIT1     TST   FLAG         SLAVE TIMER FLAG
03CE 26 FB     738:          BNE   BIT1
03D0 7A 0007   739:          DEC   FLAG         RESET FLAG
03D3 7D F602   740:          TST   PIA
03D6 27 08     741:          BEQ   ZERO
03D8 0D        742: ONE      SEC
03D9 20 01     743:          BRA   NXBIT
03DB 0C        744: ZERO     CLC
03DC 0E        745: NXBIT    CLI
03DD 39        746:          RTS
               747: ********************************************
               748: * FIND MESSAGE IDENTITY CODE:
               749: ********************************************
03DE 7F 0016   750: IDPAT    CLR   TEMP         USE TEMP FOR RECEIVED
03E1 C6 07     751:          LDA B #7           DATA.
03E3 37        752: IDPAT0   PSH B              SAVE BIT COUNTER
03E4 BD 93     753:          JSR   BIT          GET BIT
03E7 79 0016   754:          ROL   TEMP
03EA 33        755:          PUL B              DO 7 TIMES
03EB 5A        756:          DEC B
03EC 26 F6     757:          BNE   IDPAT0
03EE 96 16     758:          LDA A TEMP         GET 7-BIT CODE
03F0 B8 0872   759:          EOR A IDPT1        CORRELATE WITH 1ST
03F3 BD 22     760:          JSR   IDCOR        PATTERN
03F5 97 17     761:          STA A TEMP+1       STORE RESULT
03F7 B6 0872   762:          LDA A IDPT1
03FA 97 13     763:          STA A SNPAT
03FC 96 16     764:          LDA A TEMP         GET CODE
03FE B8 0873   765:          EOR A IDPT2        CORRELATE WITH 2ND
0401 BD 1E     766:          JSR   IDCOR
0403 91 17     767:          CMP A TEMP+1       COMPARE WITH PREVIOUS
0405 2F 07     768:          BLE   IDPAT1       IF LESS, IGNORE.
0407 97 17     769:          STA A TEMP+1       IF GREATER, SAVE.
0409 B6 0873   770:          LDA A IDPT2
040C 97 13     771:          STA A SNPAT
040E 96 16     772: IDPAT1   LDA A TEMP         GET CODE
0410 B8 0874   773:          EOR A IDPT3        CORRELATE WITH 3RD
0413 BD 22     774:          JSR   IDCOR
0415 91 17     775:          CMP A TEMP+1       COMPARE WITH PREVIOUS
0417 2F 05     776:          BLE   IDPAT2       IF LESS, IGNORE
0419 B6 0874   777:          LDA A IDPT3
041C 97 13     778:          STA A SNPAT
041E 96 13     779: IDPAT2   LDA A SNPAT        RETURN PATTERN
```



```
0420 39        780:          RTS
               781: *
               782: * CORRELATE 7 BITS:
               783: *
0421 C6 07     784: IDCOR    LDA B #7
0423 D7 19     785:          STA B SNPAT+1      USE THIS BYTE AS COUNTER
0425 5F        786:          CLR B
0426 44        787: IDCR1    LSR A
0427 25 01     788:          BCS   IDCR2
0429 5C        789:          INC B
042A 7A 0019   790: IDCR2    DEC   SNPAT+1
042D 26 F7     791:          BNE   IDCR1
042F 17        792:          TBA                CORR COEFF IN ACCA
0430 39        793:          RTS
               794: *
               795: * MESSAGE IDENTITY PATTERNS:
               796: * (SHIFTED 7-BIT M-SEQUENCES)
               797: *
0431 74        798: IDPT1    FCB   %01110100
0432 53        799: IDPT2    FCB   %01010011
0433 4E        800: IDPT3    FCB   %01001110
               801: ********************************************
               802: * CORRELATE 8 BITS IN ACCA WITH BYTE POINTED
               803: * TO BY ERRPTR. (DATA RIGHT JUSTIFIED)
               804: ********************************************
0434 36        805: ERROR    PSH A              SAVE ACCA
0435 DF 06     806:          STX   XTEMP
0437 DE 0D     807:          LDX   ERRPTR
0439 A8 00     808:          EOR A X
043B 37        809:          PSH B              BIT COUNTER
043C C1 08     810: ERR0     CMP B #8
043E 27 04     811:          BEQ   ERR1
0440 48        812:          ASL A              LEFT JUSTIFY RESULT
0441 5C        813:          INC B
0442 20 F8     814:          BRA   ERR0
0444 33        815: ERR1     PUL B
0445 49        816: ERR3     ROL A
0446 24 05     817:          BCC   BITOK
0448 BD 049B   818:          JSR   BITERR       BIT IN ERROR
044B 20 03     819:          BRA   ERR2
044D BD 08DB   820: BITOK    JSR   NBITER       BIT NOT IN ERROR
0450 5A        821: ERR2     DEC B
0451 26 F2     822:          BNE   ERR3
0453 08        823: ERR4     INX
0454 DF 0D     824:          STX   ERRPTR
0456 DE 06     825:          LDX   XTEMP        RESTORE XREG
0458 32        826:          PUL A              RESTORE DATA BYTE
0459 39        827:          RTS
               828: ********************************************
               829: * BIT IN ERROR:
               830: ********************************************
045A 36        831: BITERR   PSH A              SAVE REGISTERS
045B 37        832:          PSH B
045C DF 14     833:          STX   XTEMP2
045E DE 0F     834:          LDX   ITPTR
0460 7D 000C   835:          TST   PREV         ERROR ON PREVIOUS BIT ?
0463 26 0E     836:          BNE   BITER1
0465 08        837:          INX                NO: START A NEW COUNTER
0466 08        838:          INX
0467 4F        839:          CLR A
```

```
0889 A7 00      340:        STA A X
088B 4C         341:        INC A
088C A7 01      342:        STA A 1,X           START OFF WITH 1
088E 86 FF      343:        LDA A #$FF          END OF BUFFER MARKER
0890 A7 02      344:        STA A 2,X
0892 20 0E      345:        BRA   BITER2
0894 A6 01      346: BITER1 LDA A 1,X           ERROR ON PREVIOUS BIT
0896 8B 01      347:        ADD A #1            SO BUMP CURRENT COUNTER
0898 19         348:        DAA
0899 A7 01      349:        STA A 1,X
089B A6 00      350:        LDA A X
089D 89 00      351:        ADC A #0
089F 19         352:        DAA
08A0 A7 00      353:        STA A X
08A2 DF 0F      354: BITER2 STX   SITPTR         NEW POINTER
08A4 96 05      355:        LDA A ERRCNT+1      BUMP TOTAL ERROR COUNT
08A6 8B 01      356:        ADD A #1
08A8 19         357:        DAA
08A9 97 0B      358:        STA A ERRCNT+1
08AB 96 0A      359:        LDA A ERRCNT
08AD 89 00      360:        ADC A #0
08AF 19         361:        DAA
08B0 97 0A      362:        STA A ERRCNT
08B2 86 FF      363:        LDA A #$FF          INDICATE ERROR
08B4 97 0C      364:        STA A PREV
08B6 DE 14      365:        LDX   XTEMP2        RESTORE REGISTERS
08B8 33         366:        PUL B
08B9 32         367:        PUL A
08BA 39         368:        RTS
                369: ********************************************
                370: * BIT NOT IN ERROR:
                371: ********************************************
08BB 36         372: NBITEP PSH A
08BC 37         373:        PSH B
08BD DF 14      374:        STX   XTEMP2        SAVE REGISTERS
08BF DE 0F      375:        LDX   SITPTR
08C1 7D 0630    376:        TST   PREV          ERROR ON PREVIOUS BIT ?
08C4 27 0E      377:        BEQ   NBITR1
08C6 08         378:        INX                 YES; START A NEW COUNTER
08C7 08         379:        INX
08C8 4F         380:        CLR A
08C9 A7 00      381:        STA A X
08CB 4C         382:        INC A
08CC A7 01      383:        STA A 1,X
08CE 86 FF      384:        LDA A #$FF
08D0 A7 02      385:        STA A 2,X
08D2 20 0E      386:        BRA   NBITR2
08D4 A6 01      387: NBITR1 LDA A 1,X           NO ERROR ON PREVIOUS
08D6 8B 01      388:        ADD A #1
08D8 19         389:        DAA
08D9 A7 01      390:        STA A 1,X
08DB A6 00      391:        LDA A X
08DD 89 00      392:        ADC A #0
08DF 19         393:        DAA
08E0 A7 00      394:        STA A X
08E2 DF 0F      395: NBITR2 STX   SITPTR
08E4 4F         396:        CLR A
08E5 97 0C      397:        STA A PREV
08E7 DE 14      398:        LDX   XTEMP2
08E9 33         399:        PUL B
```



```
090A 32         900:        PUL A
090B 39         901:        RTS
                902: ********************************************
                903: * REAL TIME CLOCK
                904: ********************************************
090C DE 04      905: IRQPRC LDX   SCOUNT
090E 08         906:        INX
090F DF 04      907:        STX   SCOUNT        SECONDS COUNTER
0911 96 02      908:        LDA A SECS
0913 8B 01      909:        ADD A #1
0915 19         910:        DAA
0916 97 02      911:        STA A SECS
0918 81 60      912:        CMP A #$60
091A 26 22      913:        BNE   IRQRTN
091C 7F 0002    914:        CLR   SECS
091F 96 01      915:        LDA A MINS
0921 8B 01      916:        ADD A #1
0923 19         917:        DAA
0924 97 01      918:        STA A MINS
0926 81 60      919:        CMP A #$60
0928 26 14      920:        BNE   IRQRTN
092A 7F 0001    921:        CLR   MINS
092D 96 00      922:        LDA A HOURS
092F 8B 01      923:        ADD A #1
0931 19         924:        DAA
0932 97 00      925:        STA A HOURS
0934 81 24      926:        CMP A #$24
0936 26 06      927:        BNE   IRQRTN
0938 7F 0000    928:        CLR   HOURS
093B 7C 0003    929:        INC   DAY
093E B6 F530    930: IRQRTN LDA A PIAT
0941 3B         931:        RTI
                932: ********************************************
                933: * FETCH TIME
                934: ********************************************
0942 CE 071E    935: FETTIM LDX   #CRLF
0945 BD E07E    936:        JSR   PDATA
0948 CE 0975    937:        LDX   #TMMES
094B BD E07E    938:        JSR   PDATA
094E BD 0993    939:        JSR   DECNM
0951 97 00      940:        STA A HOURS
0953 BD E073    941:        JSR   INCH
0956 81 3A      942:        CMP A #':
0958 26 E8      943:        BNE   FETTIM
095A BD 0993    944:        JSR   DECNM
095D 97 01      945:        STA A MINS
095F BD E073    946:        JSR   INCH
0962 81 3A      947:        CMP A #':
0964 26 DC      948:        BNE   FETTIM
0966 BD 0993    949:        JSR   DECNM
0969 97 02      950:        STA A SECS
096B CE 071E    951:        LDX   #CRLF
096E BD E07E    952:        JSR   PDATA
0971 7F 0003    953:        CLR   DAY
0974 39         954:        RTS
0975 45         955: TMMES  FCC   'ENTER TIME OF DAY (HH:MM:SS) '
0992 04         956:        FCB   4
                957: ********************************************
                958: * FETCH DECIMAL NO.
                959: ********************************************
```

```
0893 BD E039   960: DECNM  JSR   INHEX
0896 48        961:       ASL A
0897 48        962:       ASL A
0898 48        963:       ASL A
0899 48        964:       ASL A
089A 16        965:       TAB
089B BD E039   966:       JSR   INHEX
089E 1B        967:       ABA
089F 39        968:       RTS
               969: ***********************************************
               970: * CORRELATE DATA:
               971: ***********************************************
08A0 DE 0A06   972: CORR   LDX   #BUFF        RECEIVED DATA POINTER
08A3 DF 06     973:       STX   XTEMP
08A5 C6 04     974:       LDA B #4           MESSAGE COUNTER
08A7 DE 0A13   975: L1     LDX   #DATA
08AA DF 08     976:       STX   XTEMP1
08AC DE 08     977: LOOP2 LDX   XTEMP1
08AE A6 00     978:       LDA A X
08B0 81 23     979:       CMP A #'#
08B2 27 10     980:       BEQ   ENDATA
08B4 08        981:       INX
08B5 DF 08     982:       STX   XTEMP1
08B7 DE 06     983:       LDX   XTEMP
08B9 A1 00     984:       CMP A X
08BB 27 02     985:       BEQ   NERR
08BD 20 24     986:       BRA   MSERR
08BF 08        987: NERR  INX
08C0 DF 06     988:       STX   XTEMP
08C2 20 E8     989:       BRA   LOOP2
08C4 5A        990: ENDATA DEC B
08C5 26 E0     991:       BNE   L1
08C7 39        992:       RTS
08C8 54        993: DATA  FCC   'THEQUICKBROWNFOXJUMPSOVERTHELAZY!'
               994: ***********************************************
               995: * CORRELATE 2 CHARS:
               996: ***********************************************
08E9 3C        997: MSERR  PSH B
08EA 4D 00     998:       TST A X
08EC 06 07     999:       LDA B #7
08EE 46       1000: MSERR1 ROR A
08EF 36       1001:       PSH A
08F0 24 0E    1002:       BCC   MSERR2
08F2 96 13    1003:       LDA A MERR+1
08F4 8B 01    1004:       ADD A #1
08F6 19       1005:       DAA
08F7 97 13    1006:       STA A MERR+1
08F9 96 12    1007:       LDA A MERR
08FB 89 00    1008:       ADC A #0
08FD 19       1009:       DAA
08FE 97 12    1010:       STA A MERR
0900 32       1011: MSERR2 PUL A
0901 5A       1012:       DEC B
0902 26 EA    1013:       BNE   MSERR1
0904 33       1014:       PUL B
0905 39       1015:       RTS
0906          1016: BUFF  RMB   133
0987          1017: FCB0  RMB   2
0989 00       1018:       FCB   0
0989 45       1019:       FCC   'ERRDAT000'
```



```
0993          1020:       RMB   156
082F          1021: FCB1  RMB   2
0831 01       1022:       FCB   1
0832 45       1023:       FCC   'ERRPAT000'
083B          1024:       RMB   156
08D7          1025: ERRBUF RMB  256
              1026: ***********************************************************
              1027: * SLAVE PROCESSOR ROUTINES FOR DE-INTERLEAVING AND DECODING:
              1028: *       INFLAG=0 -> NO DE-INTERLEAVING REQUIRED: DECODE DATA
              1029: *                   IN TABLE INDICATED BY TBFLAG
              1030: *       INFLAG<>0 -> DE-INTERLEAVE DATA FROM ONE OF TWO
              1031: *                   TABLES:
              1032: *        TBFLAG=0 -> DATA TO BE DE-INTERLEAVED: TBINT0
              1033: *                   DATA TO BE DECODED:       TBCOD0
              1034: *       TBFLAG<>0 -> DATA TO BE DE-INTERLEAVED: TBINT1
              1035: *                   DATA TO BE DECODED:       TBCOD1
              1036: ***********************************************************
C3FE          1037:       ORG   $C3FE
C3FE 01 12    1038:       FDB   SLAVE0-LATCH
              1039: *
              1040: * SLAVE VARIABLES:
              1041: *
C000          1042:       ORG   $C000
C000          1043: LATCH  RMB   1
0001          1044:       ORG   1
0001          1045: STFLAG RMB   1            STATUS FLAG
0002          1046: INFLAG RMB   1
0003          1047: TBFLAG RMB   1
0004          1048: ERRCOR RMB   2            NO. OF CORRECTED ERRORS
0006          1049: RX     RMB   2            RECEIVED WORD R(X)
0008          1050: STEMP  RMB   2            XREG TEMP STORE
000A          1051: STEMP1 RMB   2
000C          1052: STEMP2 RMB   2
000E          1053: STEMP3 RMB   2
0010          1054: TBDECD RMB   2
0012          1055: TBDINT RMB   2
0014          1056: M      RMB   1
0015          1057: TB1PTR RMB   2
0017          1058: TB2PTR RMB   2
0019          1059: D1     RMB   1
001A          1060: D3     RMB   1
001B          1061: SIGMA2 RMB   1
001C          1062: SHFCNT RMB   1
001D          1063: D1EXP  RMB   1
001E          1064: TEMP1  RMB   1
001F          1065: I      RMB   1
0020          1066: TBINT0 RMB   30
003E          1067: TBINT1 RMB   30
005C          1068: TBCOD0 RMB   32
007C          1069: TBCOD1 RMB   32
              1070: *
              1071: * SLAVE MAIN PROCEDURE:
              1072: *
C100          1073:       ORG   $C100
C100 01       1074: TBL1   FCB   1,2,4,8,3,6,$C GALOIS FIELD LOG TABLE
C107 08       1075:       FCB   $8,5,$A,7,$E,$F,$D,9,1
C110 98       1076: M1     FCB   %10011000   MINIMUM POLYNOMIAL M1(X)
C111 F8       1077: M3     FCB   %11111000   MINIMUM POLYNOMIAL M3(X)
C112 8E 00FE  1078: SLAVE0 LDS   #$00FE      STACK POINTER
C115 7D 0003  1079:       TST   TBFLAG       WHICH TABLES REQUIRED ?
```

```
C118 27 05    1080:          BEQ    SLAVE1
C11A CE 0070  1081:          LDX    #TBCOD1      IF <> 0 THEN SELECT
C11D 20 03    1082:          BRA    SLAVE2
C11F CE 0050  1083: SLAVE1   LDX    #TBCOD0      OTHERWISE SELECT
C122 DF 10    1084: SLAVE2   STX    TBDECD       TBCOD0
C124 75 0002  1085:          TST    INFLAG       DE-INTERLEAVING REQUIRED ?
C127 27 11    1086:          BEQ    SLAVE3       SKIP NEXT SECTION IF NOT
C129 7D 0002  1087:          TST    TBFLAG       TEST TABLE FLAG AGAIN
C12C 27 05    1088:          BEQ    SLAV20
C12E CE 005E  1089:          LDX    #TBINT1
C131 20 03    1090:          BRA    SLAV21
C133 CE 0020  1091: SLAV20   LDX    #TBINT0
C136 DF 12    1092: SLAV21   STX    TBDINT
C138 8D 07    1093:          BSR    DINTLV       DE-INTERLEAVE DATA
C13A 8D 45    1094: SLAVE3   BSR    DECODE       DECODE DATA
C13C 7F 0001  1095:          CLR    ITFLAG       TELL MASTER WE'RE DONE
C13F 20 FE    1096: LOOP     BRA    LOOP
              1097: ********************************************
              1098: * DE-INTERLEAVE 30 BYTES OF DATA FROM ADDRESS CONTAINED
              1099: * IN TBDINT, AND ENTER AS (7-8) BITS IN TABLE WHOSE
              1100: * ADDRESS IS CONTAINED IN TBDECD:
              1101: ********************************************
C141 DE 12    1102: DINTLV LDX    TBDINT       ADDRESS OF INTERLEAVED BLOCK
C143 DF 0C    1103:          STX    ITEMP2
C145 C6 07    1104:          LDA B  #7           DO INFORMATION BITS
C147 DE 10    1105:          LDX    TBDECD
C149 DF 08    1106:          STX    ITEMP
C14B 8D 0A    1107:          BSR    DINTL1
C14D C6 08    1108:          LDA B  #8           DO PARITY CHECK BITS
C14F DE 10    1109:          LDX    TBDECD
C151 08       1110:          INX
C152 DF 08    1111:          STX    ITEMP
C154 8D 01    1112:          BSR    DINTL1
C156 39       1113:          RTS
              1114: ********************************************
C157 37       1115: DINTL1 PSH B
C158 DE 08    1116:          LDX    ITEMP
C15A DF 0A    1117:          STX    ITEMP1
C15C 8D 05    1118:          BSR    DINTL2
C15E 33       1119:          PUL B
C15F 5A       1120:          DEC B
C160 26 F5    1121:          BNE    DINTL1
C162 39       1122:          RTS
              1123: ********************************************
C163 C6 02    1124: DINTL2 LDA B  #2           INTERLEAVE DEPTH = 2 * 8
C165 37       1125: DINTL3 PSH B
C166 C6 08    1126:          LDA B  #8
C168 DE 0C    1127: DINTL4 LDX    ITEMP2
C16A 69 00    1128:          ROL    X            EXTRACT A BIT
C16C DE 0A    1129:          LDX    ITEMP1       AND ENTER INTO
C16E 69 00    1130:          ROL    X            NEW TABLE
C170 08       1131:          INX
C171 08       1132:          INX
C172 DF 0A    1133:          STX    ITEMP1
C174 5A       1134:          DEC B               BIT COUNTER
C175 26 F1    1135:          BNE    DINTL4
C177 DE 0C    1136:          LDX    ITEMP2
C179 08       1137:          INX
C17A DF 0C    1138:          STX    ITEMP2
C17C 33       1139:          PUL B
```



```
C17D 5A       1140:          DEC B
C17E 26 E5    1141:          BNE    DINTL3
C180 39       1142:          RTS
              1143: ********************************************
              1144: * DECODE WORDS IN SELECTED TABLE
              1145: ********************************************
C181 DE 10    1146: DECODE LDX    TBDECD       GET SELECTED TABLE ADDRESS
C183 DF 08    1147:          STX    ITEMP        INTO XTEMP
C185 DF 0A    1148:          STX    ITEMP1
C187 C6 10    1149:          LDA B  #16
C189 37       1150: DCD1   PSH B
C18A DE 08    1151:          LDX    ITEMP
C18C A6 00    1152:          LDA A  X            GET INFO BITS
C18E 84 7F    1153:          AND A  #$7F         MAKE SURE ITS ASCII
C190 E6 01    1154:          LDA B  1,X          GET CHECKBITS
C192 59       1155:          ROL B               LEFT-JUSTIFY CODEWORD
C193 49       1156:          ROL A
C194 97 06    1157:          STA A  RX           PUT IN DECODING PARAMETERS
C196 D7 07    1158:          STA B  RX+1
C198 08       1159:          INX
C199 08       1160:          INX
C19A DF 08    1161:          STX    ITEMP        READY FOR NEXT
C19C 8D 0F    1162:          BSR    DECOD        FIND CORRECT WORD
C19E DE 0A    1163:          LDX    ITEMP1
C1A0 96 06    1164:          LDA A  RX
C1A2 44       1165:          LSR A               CORRECTED CHARACTER
C1A3 A7 00    1166:          STA A  X
C1A5 08       1167:          INX
C1A6 DF 0A    1168:          STX    ITEMP1
C1A8 33       1169:          PUL B               DONE 16 ?
C1A9 5A       1170:          DEC B
C1AA 26 DD    1171:          BNE    DCD1
C1AC 39       1172:          RTS
              1173: ********************************************
              1174: * GALOIS FIELD LOGARITHM/ ANTI-LOGARITHM TABLES:
              1175: ********************************************
              1176: ************************************************
              1177: *  BEGIN DECODING PROCEDURE                     *
              1178: ************************************************
C1AD CE 0100  1179: DECOD  LDX    #TBL1-LATCH
C1B0 DF 15    1180:          STX    TB1PTR       INITIALIZE LOOKUP POINTERS
C1B2 CE 0200  1181:          LDX    #TBL2-LATCH
C1B5 DF 17    1182:          STX    TB2PTR
              1183: ************************************************
              1184: *  FIND POWER SUM SYMMETRIC FUNCTIONS S1 & S3   *
              1185: ************************************************
              1186: ********************************************
              1187: * DIVISION ROUTINE TO FIND REM(RX(X)/M)      *
              1188: ********************************************
C1B7 C6 0B    1189: SYND   LDA B  #11          INITIALIZE SHIFT COUNT
C1B9 D7 1C    1190:          STA B  SHFCNT
C1BB 96 06    1191:          LDA A  RX           GET RECEIVED WORD MOD
C1BD D6 07    1192:          LDA B  RX+1         G(X)
C1BF 88 0110  1193: DIV1   EOR A  M1-LATCH
C1C2 85 80    1194: TST1   BIT A  #$80          LEFT JUSTIFIED ?
C1C4 26 F9    1195:          BNE    DIV1         YES: EXOR AGAIN
C1C6 58       1196:          ASL B               NO: SHIFT DIVIDEND LEFT
C1C7 49       1197:          ROL A
C1C8 7A 001C  1198:          DEC    SHFCNT
C1CB 26 F5    1199:          BNE    TST1
```

```
0100 44       1200:        LSR A                RIGHT JUSTIFY REMAINDER
0101 44       1201:        LSR A
0102 44       1202:        LSR A
0103 44       1203:        LSR A
0104 97 19    1204:        STA A S1
              1205: ************************************************
              1206: * COMPUTE S3 BY USING LOOKUP TABLES TO SUM      *
              1207: * POWERS OF R(ALPHA**3).                        *
              1208: ************************************************
0106 7F 001A  1209: SYND3  CLR   S3             ZERO S3
0109 C6 0E    1210:        LDA B #14            EXPONENT COUNT
010B D7 1C    1211:        STA B SHFCNT
010D B6 06    1212:        LDA A RX             GET WORD
010F D6 07    1213:        LDA B RX+1
0111 85 20    1214: SYND30 BIT A #$80           LEFT JUSTIFIED ?
0113 26 3E    1215:        BNE   FELMNT         YES: FIND FIELD ELEMENT
0115 58       1216: SYND31 ASL B                SHIFT WORD
0116 49       1217:        ROL A
0117 7A 001C  1218:        DEC   SHFCNT         NEXT EXPONENT
011A 2C F5    1219:        BGE   SYND30         BRANCH IF >=0
011C 7D 0019  1220:        TST   S1             S1=0?
011F 26 3F    1221:        BNE   SIG2
0121 7D 001A  1222:        TST   S3             S3=0?
0124 26 1A    1223:        BNE   SIG2
0126 39       1224: NOERR  RTS                  ERROR-FREE WORD
              1225:        ORG   $C200
C200 00       1226: TBL2   FCB   0,0,1,4,2,8,5,$A,3 GALOIS FIELD ANTILOGS
C209 0E       1227:        FCB   $E,9,7,6,$D,$B,$C
C210 37       1228: FELMNT PSH B                SAVE ACCS ON STACK
0211 36       1229:        PSH A
0212 D6 1C    1230:        LDA B SHFCNT         GET CURRENT EXPONENT
0214 17       1231:        TBA
0215 48       1232:        ASL A                MULTIPLY BY 3
0216 1B       1233:        ABA
0217 81 0F    1234: SYND32 CMP A #15            ADD MODULO-15
0219 2D 04    1235:        BLT   SYND33
021B 80 0F    1236:        SUB A #15
021D 20 F8    1237:        BRA   SYND32         GO TEST AGAIN
021F 97 16    1238: SYND33 STA A TB1PTR+1       SAVE EXP IN PTR
0221 DE 15    1239:        LDX   TB1PTR         GET POINTER
0223 A6 00    1240:        LDA A X              GET FIELD ELEMENT
0225 98 1A    1241:        EOR A S3             ADD IN TO S3
0227 97 1A    1242:        STA A S3             AND RETURN NEW VALUE
0229 32       1243:        PUL A                RESTORE STACK
022A 33       1244:        PUL B
022B 20 B5    1245:        BRA   SYND31
              1246: ************************************************
              1247: * FIND SIGMA2=S1*S1+S3/S1   (GF(2**4))         *
              1248: ************************************************
022D 96 19    1249: SIG2   LDA A S1             GET S1
022F 8D 10    1250:        BSR   GFSQR          FIND S1*S1
0231 97 1E    1251:        STA A TEMP1          SAVE RESULT
0233 96 1A    1252:        LDA A S3
0235 27 04    1253:        BEQ   S3ZERO         BRANCH IF ZERO
0237 D6 19    1254:        LDA B S1
0239 8D 1C    1255:        BSR   GFDIV          FIND S3/S1
023B 98 1E    1256: S3ZERO EOR A TEMP1          S1*S1+S3/S1
023D 97 1B    1257:        STA A SIGMA2
023F 20 8E    1258:        BRA   CONT           CONTINUE WITH MAIN PROCEDURE
```



```
              1260: * GALOIS FIELD SQUARING ROUTINE:              *
              1261: *  FINDS ACCA=ACCA*ACCA IN GF(2**4)           *
              1262: ************************************************
0241 97 19    1263: GFSQR  STA A TB2PTR+1       FIELD ELEMENT >>POINTER
0243 DE 17    1264:        LDX   TB2PTR         FETCH POINTER
0245 A6 00    1265:        LDA A X              FETCH EXPONENT OF ROOT
0247 97 1D    1266:        STA A S1EXP          SAVE EXPONENT
0249 48       1267:        ASL A                MULTIPLY BY 2
024A 81 0F    1268:        CMP A #15            MODULO-15
024C 2D 02    1269:        BLT   SQ1
024E 80 0F    1270:        SUB A #15
0250 97 16    1271: SQ1    STA A TB1PTR+1       NEW EXPONENT >>POINTER
0252 DE 15    1272:        LDX   TB1PTR         FETCH POINTER
0254 A6 00    1273:        LDA A X              FETCH NEW FIELD ELEMENT
0256 39       1274:        RTS
              1275: ************************************************
              1276: * GALOIS FIELD DIVISION ROUTINE:              *
              1277: *   FINDS ACCA = ACCA/ACCB IN GF(2**4)        *
              1278: ************************************************
0257 97 18    1279: GFDIV  STA A TB2PTR+1       FIELD ELEMENT >>POINTER
0259 DE 17    1280:        LDX   TB2PTR         FETCH POINTER
025B A6 00    1281:        LDA A X              FETCH EXPONENT
025D D7 18    1282:        STA B TB2PTR+1       DIVISOR FIELD ELEMENT
025F DE 17    1283:        LDX   TB2PTR
0261 E6 00    1284:        LDA B X              DIVISOR EXPONENT
0263 10       1285:        SBA                  SUBTRACT EXPONENTS
0264 2A 02    1286:        BPL   QUOT
0266 8B 0F    1287:        ADD A #15            MODULO-15
0268 97 16    1288: QUOT   STA A TB1PTR+1       NEW EXPONENT>>POINTER
026A DE 15    1289:        LDX   TB1PTR         GET POINTER
026C A6 00    1290:        LDA A X              QUOTIENT FIELD ELEMENT
026E 39       1291:        RTS
              1292: ************************************************
              1293: * BEGIN SEARCHING FOR ROOTS                   *
              1294: ************************************************
026F 4F       1295: CONT   CLR A                CLEAR "I"
0270 97 1F    1296: TEST   STA A I              SAVE CURRENT "I"
0272 48       1297:        ASL A                EXPONENT*2
0273 81 0F    1298:        CMP A #15            MODULO-15
0275 2D 02    1299:        BLT   SQ2
0277 80 0F    1300:        SUB A #15
0279 97 16    1301: SQ2    STA A TB1PTR+1
027B DE 15    1302:        LDX   TB1PTR
027D A6 00    1303:        LDA A X              GET SQUARED FIELD ELEMENT
027F 97 1E    1304:        STA A TEMP1          AND SAVE IT
0281 D6 1D    1305:        LDA B S1EXP          GET EXPONENT OF S1
0283 DB 1F    1306:        ADD B I
0285 C1 0F    1307:        CMP B #15            MODULO-2 ADD
0287 2D 02    1308:        BLT   MUL1
0289 C0 0F    1309:        SUB B #15
028B D7 16    1310: MUL1   STA B TB1PTR+1       SAVE EXPONENT
028D DE 15    1311:        LDX   TB1PTR         GET POINTER
028F E6 00    1312:        LDA B X              GET FIELD ELEMENT
0291 D8 1E    1313:        EOR B TEMP1          FIND PARTIAL SUM (GF(2**4))
0293 D8 1B    1314:        EOR B SIGMA2         ADD IN SIGMA2
0295 27 0A    1315:        BEQ   ROOT           ROOT FOUND ?
0297 96 1F    1316: RT1    LDA A I              FETCH LOCATOR
0299 81 0E    1317:        CMP A #14
029B 27 03    1318:        BEQ   STOP5          STOP IF SEARCH DONE
029D 4C       1319:        INC A                INCREMENT LOCATOR
```

```
1295 20 D9    1320:          BRA   TEST          AND TEST AGAIN
1297 39       1321: STOP5    RTS                 SEARCH COMPLETE
1298 5F       1322: ROOT     CLR B               CLEAR SHIFT COUNTER
1299 74 0006  1323:          LSR   RX            RIGHT JUSTIFY R(X)
129C 76 0007  1324:          ROR   RX+1
129F D1 1F    1325: TST2     CMP B I             IS SHIFT COUNT = I?
12A1 27 0F    1326:          BEQ   BTSWAP        YES: CORRECT ERRONEOUS BIT
12A3 06       1327: ROOT1    TAP                 GET CARRY BIT
12A4 76 0006  1328:          ROR   RX            ROTATE R(X) RIGHT
12A7 76 0007  1329:          ROR   RX+1
12AA 07       1330:          TPA                 SAVE CARRY
12AB 5C       1331:          INC B               BUMP COUNTER
12AC C1 10    1332:          CMP B #16           16 SHIFTS DONE ?
12AE 27 DE    1333:          BEQ   RTI           YES: EXIT
12B0 20 ED    1334:          BRA   TST2          LOOP AGAIN
12B2 37       1335: BTSWAP   PSH B               SAVE COUNTER ON STACK
12B3 D6 07    1336:          LDA B RX+1
12B5 C8 01    1337:          EOR B #1            SWITCH THE BIT IN ERROR
12B7 D7 07    1338:          STA B RX+1          AND RESTORE R(X)
12B9 BD 03    1339:          JSR   ERREND        BUMP CORRECTION COUNTER
12BC 33       1340:          PUL B               RESTORE STACK
12BD 20 E5    1341:          BRA   ROOT1
              1342: ********************************************
              1343: * ERROR FOUND AND CORRECTED:
              1344: ********************************************
12BF 36       1345: ERREND   PSH A
12C0 37       1346:          PSH B
12C1 DF 0E    1347:          STX   STEMP3
12C3 96 05    1348:          LDA A ERRCOR+1
12C5 8B 01    1349:          ADD A #1
12C7 19       1350:          DAA
12C8 97 05    1351:          STA A ERRCOR+1
12CA 96 04    1352:          LDA A ERRCOR
12CC 89 00    1353:          ADC A #0
12CE 19       1354:          DAA
12CF 97 04    1355:          STA A ERRCOR
12D1 DE 0E    1356:          LDX   STEMP3
12D3 33       1357:          PUL B
12D4 32       1358:          PUL A
12D5 39       1359:          RTS
              1360: ********************************************
              1361: * END SLAVE ROUTINES:
              1362: ********************************************
0000          1363:          ORG   $0000
0000          1364: LATCH1   RMB   1
0100          1365:          ORG   $0100
0100 DE 0873  1366: SLOOP    LDX   #2160
0102 09       1367: SLOOP1   DEX
0103 26 FD    1368:          BNE   SLOOP1
0105 7F 0000  1369:          CLR   FLAG-LATCH1
0108 20 F5    1370:          BRA   SLOOP
03FE          1371:          ORG   $03FE
03FE 01 00    1372:          FDB   SLOOP-LATCH1
0400          1373:          ORG   $0400
0400          1374: FLAG     RMB   1
              1375:          END
    NO ERROR(S) DETECTED
```

APPENDIX 3

PUBLICATIONS

(1) "High Speed Software Cassette Interface for the SWTP 6800 System"

(2) "Real-time Adaptive Coding of Ionospherically Propagated Data"

(3) "Microprocessor Controlled Trackside Recorder for British Rail"

HIGH-SPEED SOFTWARE CASSETTE INTERFACE FOR THE SWTP 6800 SYSTEM

David R. Isaac
Dept. of Applied Physics & Electronics, University of Durham

This article describes a software approach to the problem of interfacing a microprocessor system (in this case the SWTP 6800 system) to an audio cassette used for data storage. This alternative to the more conventional hardware methods offers several distinct advantages. Since the system is software based (the only hardware requirements being the cassette player and two ports of a PIA), its most obvious attraction is low cost. Unlike dedicated hardware cassette interfaces, total control over speed is available and the system is capable of data transfer rates up to about 2000 baud. With the parameters given in the listings, a rate of 1600 baud is obtainable.

Figure 1 illustrates the hardware configuration. A PIA is plugged into row 4 in the motherboard and output to the mic. socket of the cassette recorder is taken directly via a screened lead from the most significant bit of the 'A' side of the PIA. Incoming data from the cassette player is received via another screened lead connected from the extension speaker socket to the most significant bit of the 'B' side. Good results were obtained using medium quality audio cassettes and with the volume and tone controls on the cassette player both set to maximum.

Data is stored on the cassette as a frequency shift keyed signal. Logic levels '0' and '1' are stored as single cycles of 2800 and 1400 Hz respectively which represents an average bit rate of 1600 baud. Data is sent as a MIKBUG formatted tape headed by a string of 200 ASCII nulls and terminated with an ASCII 'S9' end of file pattern. The 'save' and 'read' programs are shown in Figures 2 and 3 respectively, and have been given arbitrary origins, although both programs are fully relocatable. By storing the programs in PROM, the risk of overwriting them is eliminated.

To save an area of data on cassette tape, first enter the beginning address of the area in locations A002 and A003 and the final address in locations A004 and A005. Set locations A048 and A049 to the start address of the SAVE program, switch the cassette player on to record and enter G. When the transfer of data is complete, the system automatically returns to MIKBUG.

To read data from the cassette, first set locations A048 and A049 to the start address of the READ program, start the cassette rolling a little before the data begins (to allow it to pick up speed) then enter G. When the end of file pattern is read from the tape the system returns to MIKBUG. Should an error be encountered during playback, an 'E' will be printed on the teletype; the tape should be stopped, rewound slightly and a G should again be entered. Note that there is no need to reset the program counter before doing this.

This cassette system has been used for several months and has been found to be very reliable. The data transfer rate is more than five times that of Kansas City standard system which results in a considerable time saving for long programs. A change of speed can be effected by changing the frequencies of the two FSK tones. This is done by altering the timing loop parameter at location 50D0 in the SAVE program, and the mean number of samples per half-cycle at location 6008 in the READ program.

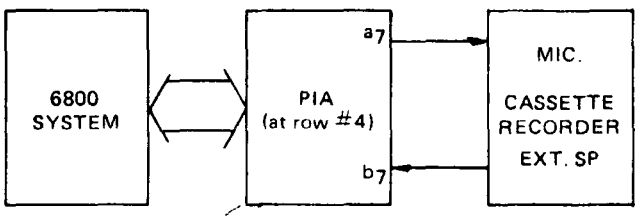


Figure 1. Hardware Configuration

```
TITLE:  SAVE

LOC   OBJECT CODE      SOURCE STATEMENTS

                            NAM    SAVE
          *
          *
          *                 TRANSMIT ROUTINES
          *
          *    This program sends mikbug formatted data as
          *    an FSK signal through one port of A Pia.
          *    A logic '1' is sent as one cycle of 1400 Hz;
          *    A logic '0' is sent as one cycle of 2800 Hz.
          *    Bega, Bega + 1 and Enda, Enda + 1 Contain the
          *    first and last addresses respectively of the
          *    data area to be saved.
          *
          *
          8010        PIAAD   EQU    $8010    PIA ADDRESS
                      * MIKBUG RAM ADDRESSES:
          A002        BEGA    EQU    $A002    DATA START ADDRESS
          A004        ENDA    EQU    $A004    DATA END   ADDRESS
          A00E        TEMP    EQU    $A00E
          A00F        TW      EQU    $A00F
          A011        MCONT   EQU    $A011
          A012        XTEMP   EQU    $A012
                      * END-OF-FILE PATTERN:
A020                          ORG    $A020
A020  0D              MTAPE2  FCB    $D,$A,0,0,0,0,$53,$39,4
A021  0A
A022  00
A023  00
A024  00
A025  00
A026  53
A027  39
A028  04
                      * MIKBUG ROM ADDRESSES
          E0E3        CONTRL  EQU    $E0E3
          E134        MTAPE1  EQU    $E134
5000                          ORG    $5000    PROGRAM ORIGIN
5000  CE FF04         START   LDX    #$FF04   INITIALISE PIA
5003  FF 8010                 STX    PIAAD
                      *  SEND 200 NULLS
5006  C6 C8                   LDA B  #200
5008  4F              NULLS   CLR A           NULL
5009  8D 67                   BSR    SEND
500B  5A                      DEC B
500C  26 FA                   BNE    NULLS
                      * MIKBUG "PUNCH" ROUTINE:
500E  FE A002         PUNCH   LDX    BEGA     START ADDRESS
5011  FF A00F                 STX    TW       SAVE IT
5014  B6 A005         SEN11   LDA A  ENDA+ 1
5017  B0 A010                 SUB A  TW+1
501A  F6 A004                 LDAB   ENDA
501D  F2 A00F                 SBCB   TW
5020  26 04                   BNE    SEN22
5022  81 10                   CMP A  #16
5024  25 02                   BCS    SEN23
5026  86 0F         SEN22     LDA A  #15
5028  8B 04         SEN23     ADD A  #4
502A  B7 A011                 STA A  MCONT   FRAME COUNT THIS RECORD
502D  80 03                   SUB A  # 3
502F  B7 A00E                 STA A  TEMP    BYTE COUNT THIS RECORD
                      * SEND  C/R, L/F, NULLS, S, 1 (START OF RECORD):
5032  CE E134                 LDX    #MTAPE1
5035  8D 14                   BSR    PDATA1
5037  5F                     CLR B           ZERO CHECKSUM
                      * SEND  FRAME COUNT
5038  CE A011                 LDX    #MCONT
503B  8D 38                   BSR    SEND2
                      * SEND  ADDRESS
503D  CE A00F                 LDX    #TW
5040  8D 33                   BSR    SEND2
5042  8D 31                   BSR    SEND2
                      * SEND  DATA
5044  FE A00F                 LDX    TW
5047  8D 2C         SEN32     BSR    SEND2   SEND ONE BYTE (2 FRAMES)
5049  7A A00E                 DEC    TEMP    DEC BYTE COUNT
504C  26 F9                   BNE    SEN32
504E  FF A00F                 STX    TW
5051  53                     COM B
5052  37                     PSH B
5053  30                     TSX
                      * SEND  CHECKSUM
5054  8D 1F                   BSR    SEND2
5056  33                     PUL B   RESTORE STACK
5057  FE A00F                 LDX    TW
505A  09                     DEX
505B  BC A004                 CPX    ENDA
505E  26 B4                   BNE    SEN11
                      * IF FINISHED, SEND END-OF-FILE
5060  CE A020                 LDX    #MTAPE2
5063  8D 06                   BSR    PDATA1
5065  7E E0E3                 JMP    CONTRL  RETURN TO MIKBUG
5068  8D 24         PDATA2    BSR    SENDIT
506A  08                     INX
506B  A6 00         PDATA1    LDA A  X
506D  81 04                   CMP A  #4
506F  26 F7                   BNE    PDATA2
5071  39                     RTS
5072  8D 1A         SEND      BSR    SENDIT
5074  39                     RTS
                      * SEND  2 HEX CHARS: UPDATE CHECKSUM
```

Continued on page 42

```
5  EB 00        SEND2   ADD A  O,X   UPDATE CHECKSUM
7  A6 00        OUT2H   LDA A  O,X   SEND 2 HEX CHARS
9  8D 05        OUT2HA  BSR    OUTHL  OUT LEFT CHAR
B  A6 00                LDA A  O,X
D  08                   INX
E  20 04                BRA    OUTHR  OUT RIGHT CHAR & RETURN
0  44           OUTHL   LSR A
1  44                   LSR A
2  44                   LSR A
3  44                   LSR A
4  84 0F        OUTHR   AND A  #$F    OUT RIGHT BCD DIGIT
6  8B 30                ADD A  #$30
8  81 39                CMP A  #$39
A  23 02                BLS    SENDIT
C  8B 07                ADD A  #$7
                *
                * GENERATE FSK SIGNAL AND SEND THROUGH PIA
                *
E  37           SENDIT  PSH B
F  FF A012              STX    XTEMP  SAVE XREG
2  C6 08                LDA B  #8     BIT COUNT
4  8D 29                BSR    ZERO   SEND START BIT
6  0D                   SEC
7  49           NEXBIT  ROL A         GET NEXT BIT
8  24 04                BCC    SKIP   JUMP IF ZERO
A  8D 14                BSR    ONE    SEND A '1'
C  20 02                BRA    OVER
E  8D 1F        SKIP    BSR    ZERO   SEND A '0'
0  0D           OVER    SEC
1  5A                   DEC B
2  26 F3                BNE    NEXBIT NEXT BIT
4  8D 0A                BSR    ONE    SEND STOP BIT
6  86 00                LDA A  #0     PIA---> LOW
8  B7 8010              STA A  PIAAD
B  FE A012              LDX    XTEMP  RESTORE XREG
E  32                   PUL A         AND STACK
F  39                   RTS
                * BIT=1; TRANSMIT CYCLE OF LOW FREQUENCY
0  36           ONE     PSH A         SAVE ACCA
1  86 00        AGAIN   LDA A  # 0    PIA---> LOW
3  8D 15                BSR    DELAY  TIMER
5  8D 13                BSR    DELAY  AND AGAIN
```

```
50B7  86 FF              LDA A  #$FF   PIA--> HIGH
50B9  8D 0F              BSR    DELAY
50BB  8D 0D              BSR    DELAY  WAIT AGAIN
50BD  32                 PUL A         RESTORE STACK
50BE  39                 RTS
                * BIT=0; TRANSMIT CYCLE OF HIGH FREQUENCY
50BF  36        ZERO     PSH  A        SAVE ACCA
50C0  86 00     BACK     LDA A  #0     PIA--> LOW
50C2  8D 06              BSR    DELAY  TIMER
50C4  86 FF              LDA A  #$FF   PIA--> HIGH
50C6  8D 02              BSR    DELAY
50C8  32                 PUL A         RESTORE STACK
50C9  39                 RTS
                * FREQUENCY CONTROL DELAY LOOP
50CA  37        DELAY    PSH B         SAVE ACCB
50CB  C6 08              LDA B  #8     DELAY COUNTER
50CD  B7 8010   LOOP1    STA A  PIAAD  OUTPUT TO PIA
50D0  5A                 DEC B         DEC LOOP COUNT
50D1  26 FA              BNE    LOOP1  LOOP AGAIN
50D3  33                 PUL B         RESTORE STACK
50D4  39                 RTS           RETURN
                         END
```

SYMBOL	VALUE		
		OVER	50A0
AGAIN	50B1	PDATA1	506B
BACK	50C0	PDATA2	5068
BEGA	A002	PIAAD	8010
CONTRL	E0E3	PUNCH	500E
DELAY	50CA	SEND	5072
ENDA	A004	SENDIT	50BE
LOOP1	50CD	SEND2	5075
MCONT	A011	SEN11	5014
MTAPE1	E134	SEN22	5026
MTAPE2	A020	SEN23	5028
NEXBIT	5097	SEN32	5047
NULLS	5008	SKIP	509E
ONE	50B0	START	5000
OUTHL	5080	TEMP	A00E
OUTHR	5084	TW	A00F
OUT2H	5077	XTEMP	A012
OUT2HA	5079	ZERO	50BF

```
ITLE:  READ

OC   OBJECT CODE        SOURCE STATEMENTS

                          NAM       READ
        *
        *
        *                 RECEIVE ROUTINES
        *
        *     This program demodulates the fsk signal from
        *     the cassette player by comparing the number
        *     of samples per 1/2 cycle with a threshold (mean)
        *     value.   The received data is stored directly
        *     into ram.   If a non-hex char is received or
        *     a checksum error is detected, an 'E' is typed
        *     on the terminal before returning to mikbug.
        *     A return to mikbug without the 'E' indicates
        *     that all data has been successfully transferred.
        *
        *
     8012       PIAAD   EQU   $8012     PIA ADDRESS
                * MIKBUG RAM ADDRESSES:
     A00A       CKSM    EQU   $A00A
     A00B       BYTECT  EQU   $A00B
     A00C       XHI     EQU   $A00C
     A00D       XLOW    EQU   $A00D
     A012       XTEMP   EQU   $A012
     A020       CNTR    EQU   $A020
     A021       MEAN    EQU   $A021
                * MIKBUG ROM ADDRESSES
     E0E3       CONTRL  EQU   $E0E3
     E1D1       OUTCH   EQU   $E1D1
00                  ORG   $6000     PROGRAM ORIGIN
00   CE 0004        LDX   #$0004    INITIALISE PIA
03   FF 8012        STX   PIAAD
                * SET MEAN NO. OF SAMPLES PER 1/2 CYCLE:
06   86 0B          LDA A #$0B
08   B7 A021        STA A MEAN
                * BEGIN SEARCHING FOR DATA:
0B   8D 68     LOAD3   BSR   READ1     FETCH A CHAR
0D   81 53             CMP A #'S       IS IT AN S ?
0F   26 FA             BNE   LOAD3     NO;LOOK AGAIN
11   8D 62             BSR   READ1     YES; GET NEXT CHAR
13   81 39             CMP A #'9       END-OF-FILE ?
15   27 26             BEQ   LOAD21    YES; READING COMPLETE
17   81 31             CMP A #'1       START OF RECORD ?
19   26 F0             BNE   LOAD3     NO; LOOK AGAIN
1B   7F A00A           CLR   CKSM      ZERO CHECKSUM
1E   8D 2E             BSR   BYTE      READ 1 BYTE
20   80 02             SUB A #2
22   B7 A00B           STA A BYTECT
                * BUILD ADDRESS:
25   8D 19             BSR   BADDR
                * STORE DATA:
27   8D 25     LOAD11  BSR   BYTE
29   7A A00B           DEC   BYTECT
2C   27 05             BEQ     LOAD15
2E   A7 00             STA A   X       STORE DATA
30   08                INX
31   20 F4             BRA     LOAD11  GET NEXT BYTE
33   7C A00A   LOAD15  INC     CKSM
36   27 D3             BEQ     LOAD3
                * ERROR DETECTED:
38   86 45     LOAD19  LDA A   #'E     PRINT ERROR MESSAGE
```

```
603A  BD E1D1             JSR     OUTCH
603D  7E E0E3     LOAD21  JMP     CONTRL  RETURN TO MIKBUG
                * BUILD ADDRESS:
6040  8D 0C      BADDR   BSR     BYTE    GET A BYTE
6042  B7 A00C             STA A   XHI     MSBYTE
6045  8D 07              BSR     BYTE    GET A BYTE
6047  B7 A00D             STA A   XLOW    LSBYTE
604A  FE A00C             LDX     XHI     ADDRESS WE BUILT
604D  39                  RTS
                * INPUT 1 BYTE (2 FRAMES):
604E  8D 10      BYTE     BSR     INHEX   READ HEX CHAR
6050  48                  ASL A
6051  48                  ASL A
6052  48                  ASL A
6053  48                  ASL A
6054  16                  TAB
6055  8D 09               BSR     INHEX   READ HEX CHAR
6057  1B                  ABA
6058  16                  TAB
6059  FB A00A             ADD B   CKSM
605C  F7 A00A             STA B   CKSM
605F  39                  RTS
                * INPUT HEX CHAR:
6060  8D 13      INHEX    BSR     READ1
6062  80 30               SUB A   #$30
6064  2B D2               BMI     LOAD19  NOT HEX; ERROR ?
6066  81 09               CMP A   #$09
6068  2F 0A               BLE     IN1HG
606A  81 11               CMP A   #$11
606C  2B CA               BMI     LOAD19  NOT HEX; ERROR ?
606E  81 16               CMP A   #$16
6070  2E C6               BGT     LOAD19  NOT HEX; ERROR ?
6072  80 07               SUB A   #7
6074  39        IN1HG    RTS
                * SAMPLING AND TIMING ROUTINES:
6075  37        READ1    PSH B           SAVE ACCB
6076  FF A012            STX     XTEMP   AND XREG
6079  4F        START    CLR A
607A  5F                 CLR B
607B  0D                 SEC
607C  8D 13      SBIT     BSR     LOOK    LOOK FOR START BIT
607E  25 FC              BCS     SBIT    NOT FOUND; LOOK AGAIN
6080  36                 PSH A           SAVE ACCS
6081  37        NEXBIT   PSH B
6082  8D 0D              BSR     LOOK
6084  33                 PUL B           RESTORE ACCS
6085  32                 PUL A
6086  49                 ROL A           GET RECEIVED BIT
6087  5C                 INC B           8 BITS FOUND ?
6088  C1 08              CMP B   #8
608A  26 F4              BNE     NEXBIT  NO; FETCH NEXT BIT
608C  FE A012            LDX     XTEMP   RESTORE INDEX REG
608F  33                 PUL B
6090  39                 RTS
                * FETCH A BIT:
6091  7F A020   LOOK     CLR     CNTR    SAMPLE COUNT
6094  86 80              LDA A   #$80    SELECT INPUT LINE
6096  B1 8012   LOOP1    CMP A   PIAAD   START OF CYCLE ?
6099  26 FB              BNE     LOOP1   NO; LOOK AGAIN
609B  7C A020   LOOP2    INC     CNTR    YES; INC SAMPLE COUNT
609E  01                 NOP
609F  B1 8012            CMP A   PIAAD   INPUT STILL HIGH ?
60A2  27 F7              BEQ     LOOP2   YES; KEEP LOOPING.
60A4  B6 A020            LDA A   CNTR    TOTAL SAMPLE COUNT
60A7  B1 A021            CMP A   MEAN    COMPARE WITH MEAN
```

Pg. 43

```
60AA 2D 02                  BLT   HP      IF LESS THEN HI FREQUENCY
60AC 0D                LP   SEC           RETURN '1' IN CARRY
60AD 39                     RTS
60AE 0C                HP   CLC           RETURN '0' IN CARRY
60AF 39                     RTS
                            END

SYMBOL          VALUE  CKSM  A00A    INHEX   6060    LOOP2   6098  XTEMP  A012
                       CNTR  A020    LOAD3   6091    XHI     A00C
BADDR           6040   CONTRL E0E3    LOOK    6096    XLOW    A00D
BYTE            604E   HP     60AE    LOOP1   6074    MEAN    A021
BYTECT          A00B   INHEX  6060    LOOP2   60AC    NEXBIT  6080
                       LOAD11         LP      6027    OUTCH   E1D1
                       LOAD15         LOAD19  6033    PIAAD   8012
                       LOAD19         LOAD21  6038    READ1   6075
                       LOAD21         LOAD3   603D    SBIT    607C
                       LOAD3          LOOK    6008    START   6079
                       LOOK           LOOP1   6091    XHI     A00C
                       LOOP1          LOOP2   6096    XLOW    A00D
                       LOOP2          INHEX   6098    XTEMP   A012
```

Real - time adaptive - coding of ionospherically propagated data

D.R. Isaac and C.T. Spracklen
Department of Applied Physics and Electronics
Durham University
ENGLAND

Abstract

High frequency radio links are time - varying channels which can introduce considerable levels of attenuation and delay distortions. These effects, together with interference from natural and man - made sources, result in the high error rates observed over the HF channel. This paper describes a novel microprocessor implementation of a data transmission system for use over HF radio, where 16 data channels are used to modulate 8 frequency - agile subcarriers contained within the voice channel. The system attempts to locate the subchannels within the quiet portion of the channel to avoid narrow band interference. Broad - band effects are overcome by using bit - interleaved block codes. The software - orientated implementation renders the system both cost - effective and flexible.

1. Introduction

Information transmitted via high frequency (HF) radio signals reflected from the ionosphere is frequently degraded due to the characteristics of the propagation medium. The high error rates encountered can be attributed to the effects of multipath propagation and interference, both natural and man - made.

This paper describes an experiment being undertaken by the Department of Applied Physics and Electronics, Durham University, intended to demonstrate that with the aid of modern electronics technology in the form of microprocessors, it is possible to effect a significant improvement in the error rate.

The simplest, and often the most effective, way of reducing these errors is to implement some form of channel evaluation system to assess the suitability of a number of channels for transmission over the radio link /1,2/, and to choose the channel yielding the best signal - to - noise ratio. This approach may often be unrealistic since a wide selection of frequencies may not be available to the communicator and he may often have to make the best possible use of a frequency far removed from the optimum. However, even for a single assigned frequency, substantial improvements can be obtained by using adaptive techniques based on a 'microscopic' analysis of the channel /3/.

Over recent years there has been a considerable resurgence of interest in communications using the HF part of the radio spectrum. Many users faced with the problems of cost and vunerability of satellite communications, have begun to look again at the prospect of achieving reliable data communications using HF radio links. Indeed, the HF path is the only alternative for long distance circuits involving mobiles such as ships and aircraft. However, the problems are considerable. If we consider signals propagated to outside the skip zone (where ionospheric propagation predominates) then we have the difficulty of communicating via an anisotropic time - varying medium with noise levels greatly exceeding those found in other communications systems. However, there has been considerable progress in the state of electronics technology in recent years, particularly in the digital field, and it seems likely that such a communications system would need to take full advantage of such advances.

2. Design Considerations

Multipath propagation causes time dispersion of the received signal /4/ which results in severe intersymbol interference if the signal element duration is of insufficient length. It has been shown /5/ that the optimum frame length for transmission over a dispersive medium is equal to $\sqrt{L/B}$, where L is the time spread introduced by the medium and B is the frequency spread. The time spread caused by multipath propagation, and the Doppler shift introduced by ionospheric perturbations, result in a near - optimum frame rate of 75 Hz. Serial data transmission systems are therefore limited to a data rate of 75 baud, which makes inefficient use of the 3 kHz. voice channel. To utilise the channel more efficiently the data can be time division multiplexed for transmission over a number of parallel sub - channels, orthogonally spaced within the voice channel.

Several such parallel sub - channel modems have been developed in the past, notably Kineplex /6/, Kathryn /7,8/, and Codem /9/, for medium speed data transmission over HF radio /10,11/.

A macroscopic investigation of the HF spectrum below the M.U.F. /1,12/ reveals that it is very difficult to find a 3 kHz. slot which is completely free of interference and it has also been shown /3/ that much of this interference is narrow - band. It is therefore unwise to attempt to use the channel to its full capacity. Codem uses a spectral redundancy technique to overcome narrow - band interference and fading by using only 16 of its 25 subchannels to carry information, the remainder being used for redundant parity bits. This is effective but inefficient in that the available transmitter power is distributed amongst all 25 channels which may present a disadvantage when working from a mobile transmitter. Gott /3/ and Betts /12/ have shown that interference measurements made on the voice channel are often valid for several minutes and occasionally for an hour or more. This suggests that an adaptive system might be used which avoids those subchannels exhibiting poor error performance. Ideally, a revertive link would be used to assess the signal - to - noise ratio on each subchannel at the receiving site by transmitting a sounding signal from the receiver back to the transmitter. In practice, revertive links are costly and often difficult to implement, especially from mobile sites, and the best that can be done is for the transmitting site to evaluate the spectral distribution of interference within the channel in the absence of a signal. It has been shown /3/ that over distances of several hundred miles the interference pattern at the transmitter site is similar to that observed at the receiver and provides a reasonable criterion for sub - channel selection. Results have also shown /13,14/ that, for a 2 - tone F.S.K. system, the error performance of an ideal frequency agile system greatly exceeds that of a system using fixed frequency allocation.

The system currently being developed uses 16 available equally (orthogonally) spaced sub - channels which accomodate 8 quartenary phase - modulated sub - carriers. One possible distribution of the sub - carriers is illustrated in Figure 1. A 4 - phase modulated signal occupies the same bandwidth as a bi - phase one but carries twice as much information, and was therefore chosen in preference /15,16/. Systems employing pilot tone phase references have been shown to exhibit poor performance on HF /17,18/, so a differential encoding technique is chosen to eliminate the need for an absolute phase reference. The differential phases correspond to dibits arranged in a

de around the unit circle (Figure 2) such that
t probable phase error causes only a single bit
Selection of the sub - channels to be used for
sion is based on the results of a measurement
noise present in each of the 16 available sub -
over an observation period of 3.5 seconds.
ure is a compromise arrived at by consideration
major factors. Firstly, an observation period
too long necessitates a large input data buffer
transmitter if overflow is to be avoided.
, a period that is too short may coincide with
fadeout of any narrow - band interfering
These fadeouts have been observed /19,20/ to
ver periods of up to 0.5 seconds. Thirdly, as
e measurements are based on the results of a
analysis using software - implemented Fast -
Transforms, sufficient time must be allowed for
be evaluated. After each observation period,
subchannels exhibiting the lowest noise levels
sen for transmission of the next message block.
g a preamble of phase reversals used to regain
synchronisation, the receiver is advised as to
sub - channels to be used for subsequent
ion of the data. This advisory sequence has a
cy level four times greater than that used for
sion of the message. Results /12/ indicate
reasonable interval between observation periods
ne order of several minutes and each message
ce consists of 3 minutes of data.

ors observed on the HF channel predominantly
bursts and it has been shown /20,21,22,23/
- interleaved binary block coding provides the
tection against these types of errors. As the
is to be implemented in real - time at the
a code must be used which is a reasonable
ise between burst - correcting ability and
ntation complexity. The (15,7) dual - error -
g binary BCH code /25/ is reasonably simple to
in real time using software /26/, and groups of
's interleaved to a depth of 16 in serial along
the 16 data sub - channels can tolerate wide -
les or noise bursts of up to 0.43 seconds. This
a block length (of 16x16x15 = 3840 bits) in
total number of 512 errors can be corrected.

nodulation of a parallel sub - channel system is
implemented using banks of narrow - band
/6/. These are often costly and are
nt in an in - band frequency agile environment
small number of filters are in use at any one
For this reason it was decided to use the
Fourier Transform (D.F.T.) for sub - channel
in and phase decoding. Because the time
to implement the F.F.T. algorithm using
exceeds the signal element duration, a
approach is adopted by evaluating the chirp -
rm algorithm using charge - coupled - device
transversal filters /27/. D.F.T. processors
ch devices are capable of evaluating a 512 -
transform in less than 6 mS.

n Implementation

requirement for system flexibility, together
falling cost of microprocessor technology,
d that a primarily 'software - based'
tation was preferable to a purely hardware -
d approach /28/. Coding / decoding and
on / demodulation are controlled almost
by the microprocessor system, changes in
rformance being effected by simply modifying
ably - level programs. This ensures that the
e costs are kept to a minimum.

A block diagram of the complete system is shown in Figure 3. At the transmitter incoming serial data is buffered and encoded by the Motorola 6800 - based microprocessor system prior to generation of the frequencies and phases required to construct the multi sub - channel baseband signal. The data is frequency division multiplexed for transmission over eight differential quadrature phase shift keyed (DQPSK) sub - channels and the resulting composite waveform is low - pass filtered to remove components above the signal band which then forms the modulating signal to the single side - band (SSB) suppressed - carrier HF transmitter. During breaks in transmission, the baseband output from the HF receiver at the transmitting site is sampled by the A/D converter and an FFT is performed by the microprocessor to determine the optimum sub - channels for subsequent transmission.

At the receiving station, samples of the filtered baseband signal are processed by an FFT processor whose output coefficients in the frequency domain are used by the microprocessor system to demodulate the composite waveform. Decoding and re - formatting of the demodulated signal then results in the output serial data stream.

For real - time operation, it was found that single processor systems at the transmitter and receiver sites were insufficient to handle all the required tasks. For this reason, a multi - processor configuration has been developed in which one or more 'slave' processor units (Figure 4) are assigned tasks by a master processor which oversees operation of the entire system. Each slave processor unit includes a single 6800 central processing unit (CPU) and 1Kbyte of read / write or random access memory (RAM), which is used to contain data and program areas. The master processor accesses the slave control lines via a 4 - bit latch located at the base address of the slave processor. Once initialised, the slave processor unit is capable of performing tasks completely independently of the master thereby increasing the overall processing power of the complete system.

The transmitter (Figure 5) uses one slave unit for generation of the multitone modulating signal. Serial data at 550 bps is accepted into the serial interface and is loaded into the input data buffer in RAM via an interrupt service routine. Groups of seven data bits are extracted from the buffer and mapped into 15 - bit BCH codewords. A group of seven data bits can be represented as a sixth order modulo - 2 polynomial d(x) which is encoded by finding

$$r(x) = x^8 d(x) + \mathrm{rem}(x^8 d(x)/g(x))$$

where g(x) is the generator polynomial

256 of the resulting codewords are assembled into a 16 x 240 bit matrix to enable interleaving to depth 16 along each of the 16 data sub - channels. On request from the slave processor, pairs of bits are extracted from the matrix and loaded into the slave RAM, each dibit corresponding to a single frequency division multiplexed sub - channel. For a single sub - channel, the absolute phase is determined by the result of an exclusive - OR operation on the current dibit with the immediately preceding dibit for that sub - channel, the result being used as an index on an 80 - point cosine lookup table in the slave processor - RAM. Samples for each sub - channel are then added in to one of two 80 x 12 bit multiplexed shift registers; the complete waveform for a single element is thus formed after eight complete rotations. When one shift register is full, its contents are output to the low pass filter at the appropriate sample rate of 6 kHz.

A message consists of 16,384 codewords, a total transmission time of 3.4 minutes. Between messages, an estimate of the interference present in each subchannel is made by evaluating a 64 - point radix - 2 butterfly Fast Fourier Transform algorithm on the sampled data signal. The 9600 Hz. sampling frequency provides a frequency - domain resolution of 150 Hz. Selection of the sub - channels to be used for transmission of the next message sequence is made by finding the eight sub - channel slots exhibiting the lowest power spectral density averaged over eight sample periods. Transmission is then resumed by sending a preamble of phase reversals over the eight sub - carriers used for the previous message. This is followed by an advisory sequence of 8 15 - bit codewords transmitted with a four - fold spectral redundancy to ensure a high probability of correct reception. The first three information bits of each codeword relate to a sub - channel number in the range 0 to 7, the last four indicating a frequency slot in the range 0 - 15. Transfer to the new frequency slots occurs after the final element of the advisory sequence.

The receiver (Figure 6) uses one slave processor to control element synchronisation of the received signal and one for decoding of the demodulated data. Demodulation is accomplished using a charge coupled device (CCD) to evaluate the chirp - Z transform /29/. The chirp - Z transform can be derived from the expression for the discrete Fourier Transform as follows /30/:

$$F_k = \sum_{n=0}^{N-1} f_n \exp(-i2\pi nk/N) \qquad \text{(DFT)}$$

$$n,k = 0,1,2,\ldots\ldots N-1$$

using the identity $\quad 2nk = n^2 + k^2 - (k-n)^2$

we obtain

$$F_k = \sum_{n=0}^{N-1} \left[f_n \exp(-i\pi n^2/N) \exp(i\pi(k-n)^2/N) \right] \exp(-i\pi k^2/N) \qquad \text{(CZT)}$$

The implementation involves three operations

(i) the comlex sequence gn is generated by the product of fn with $\exp(-i\pi n^2/N)$

(ii) a discrete convolution is performed between gn and $\exp(i\pi(k-n)^2/N)$

(iii) the resulting sequence is multiplied by $\exp(-i\pi k^2/N)$

These steps are illustrated in Figure 7. Step (ii) involves four convolutions which are performed by the CCD transversal filters. Steps (ii) and (iii) are implemented by using multiplying digital to analog converters and lookup tables in programmable read - only memories which contain sampled sine and cosine 'chirp' waveforms.

For purely real inputs, the analysis band extends from zero to the Nyquist frequency. The CCD device with its associated circuitry implements a fixed length transform of 512 points which, for a frequency resolution of 75 Hz. requires a sampling frequency of 38.4 kHz. The total time taken to acquire 512 samples is thus 512/38400 = 13.3 mS, which is the element duration. The required signal band extends from 300 to 2550 Hz. which is available in 885 microseconds. The remaining time of 12.4 mS is thus available for processing.

Synchronisation is achieved by observing the behaviour of the phasor. For a sequence of phase reversals, the locus of the phasor as we traverse along the signal element is a spiral, and its magnitude is a triangular wave (Figure 8), whose maxima correspond to the correct sampling instants. The magnitude of the phasor can be computed from the Fast Fourier Transform by observing the Fourier coefficients at the discrete sub - channel frequencies. An estimate of the mean value of the phasor over all sub - channels can be evaluated by finding

$$|\bar{F}| = \frac{1}{8} \sum_{K_c} \sqrt{F_{K_c}^2(\mathrm{Re}) + F_{K_c}^2(\mathrm{Im})}$$

where Kc is the sample number corresponding to the sub - channel frequency. The values of $|\bar{F}|$ thus obtained indicate the position of the primary sampling instant within the frame and the slave processor then instructs an appropriate change to synchronise the frame.

The demodulated data is entered into a 480 - byte matrix in the slave decoder for de - interleaving and decoding. Peterson's decoding algorithm /31/ is used to recover the seven information bits from each received codeword. Arithmetic operations are implemented in the Galois field of (2^4) elements. Addition and subtraction are simply performed modulo - 2, while multiplication and division operations are performed using logarithmic lookup tables. The algorithm used is shown in Figure 9 and can be described as follows

If r(x) is the received word and $g(x) = m_1(x)m_3(x)$ then the two partial syndromes can be found by calculating

$$S_1 = \mathrm{rem}\,\frac{r(x)}{m_1(x)} \qquad S_3 = \mathrm{rem}\,\frac{r(x^3)}{m_1(x)}$$

If errors have occured, the syndromes will be non - zero and the elementary symmetric functions can be found directly from Newton's identities:

$$S_1 + \sigma_1 = 0$$

$$S_3 + S_2\sigma_1 + S_1\sigma_2 + \sigma_3 = 0$$

in matrix notation

$$S = M_t \Sigma$$

where

$$S = \begin{bmatrix} S_1 \\ S_3 \end{bmatrix} \qquad \Sigma = \begin{bmatrix} \sigma_1 \\ \sigma_2 \end{bmatrix} \qquad M_t = \begin{bmatrix} 1 & 0 \\ S_2 & S_1 \end{bmatrix}$$

It is then possible to find the $(\sigma_j), j=1,2$ by computing

$$\Sigma = M_t^{-1} S$$

It is then possible to solve /32/ for the error locator numbers (β_j) by substituting elements of Galois field (2^4) in the equation

$$\Sigma(x) = (x+\beta_1)(x+\beta_2)$$

$$= x^2 + \sigma_1 x + \sigma_2$$

This completes the decoding algorithm. The decoded data is stored in the output data buffer and is then converted to serial format by a parallel - to - serial converter.

nclusion

novel implementation of a multi - channel modem
ta communications over HF radio links has been
ed. The use of microprocessor technology reduces
and increases system flexibility by using a
ly software - based approach. The effects of
th distortion are overcome by using a parallel
nel system to extend the duration of the signal
s. Narrow - band interference is combated by
in - band frequency agility to locate the
iers within the quiet regions of the voice
. Because of the time - varying nature of the
l, the subcarrier locations are updated at
t intervals; thus the system adapts itself to
evailing interference pattern. Subchannel
on and demodulation are accomplished by using
coupled devices to evaluate the Discrete Fourier
rm, thereby alleviating the need for banks of
- band filters which are traditionally used in
bcarrier modems. Broad band noise effects are
ne by using bit - interleaved binary block

view of the flexibility of the system, it
e possible, if a revertive link were available,
erve fluctuations in the error patterns
ned at the receiver, and to modify the
cy in the transmitted signal to suit the types
rs received. The system then becomes fully
, the redundancy in the signal always being
icient to maintain error - free communication.
is much scope for future work!

References

n automatic HF channel monitoring system',
trell, R.A., Proc. Conf. on Advances in HF
mmunications systems, IEE, Feb. 79, pp. 57 -

se of pilot tones for real - time channel
imation of HF data circuits', Betts, J.A.;
om, R.S.; Cook, S.J.; Clark, J.G.; Proc.IEE,
122, No.9, Sept. 75.
haracteristics of Interfering signals in
onautical HF voice channels', Gott, G.F.;
niforth, M.J.D.; Proc.IEE, Vol.125, No.11, Nov.
pp. 1208 - 1212.
dium - speed digital data transmission over HF
nnels', Darnell, M.; Proc. of Conf. on
tal Processing of Signals in Communications,
ghborough, Sept. 77, pp. 393 - 402.
ximum rate waveforms for dispersive channels',
nerman, M.S.; Thomson, D.N.; General Atronics
., Philadelphia, Pa., Report. 1329 - 98 -
May 1964.
ineplex - A bandwidth - efficient binary
smission system', Mosier, R.R.; Claubough,
; AIEE Trans. on Communications and
tronics, Vol. 76, 1958, pp. 723 -728.
AN/GSC - 10 (KATHRYN) variable rate data
m for HF radio', Zimmerman, M.S.; Kirsh, A.L.;
Trans. Comm. Tech., Vol. COM - 15, Part 2,
pp. 197 - 204.
d test results of the AN/GSC - 10 (KATHRYN)
al data terminal', Kirsh, A.L.; Gray, P.R.;
, D.W.; IEEE Trans. Comm. Tech., Vol. COM
1969.
combined coding and modulation approach for
munication over dispersive channels', Chase,
EEE Trans. Comm. Tech., Vol. COM - 21,
pp. 159 - 174.
gram controlled HF modem implementation', de
, J.; Fast, D.; Michelson, A.; Carmichael,
EASCON '74, pp. 349 - 352.
Integrated HF error - controlled modem', de
, J.; Michelson, A.; GTE Sylvania, Eastern
sion, Needham Heights, Mass., 02194.
sounding and its use for control of HF (3 - 30 MHz.) adaptive systems for data transmission', Betts, J.A.; Ellington, R.P.; Jones, D.R.L.; Proc. IEE, Vol. 117, No. 12, Dec. 1970.

/13/ 'Improvement of slow - rate FSK by frequency agility and coding', Gott, G.F.; Hillam, B.; Proc. Conf. on Advances in HF Communication Systems and Techniques, Feb. 79, pp. 45 -49.

/14/ 'An HF data modem with in - band frequency agility', Darnell, M., Proc. Conf. on Advances in HF Communication Systems and Techniques, Feb. 79, pp. 50 - 54.

/15/ 'Voiceband 4 - phase data set for Global Communication', Horlacher, R.L.; Schroeder, H.C.; IEEE Trans. Com. Tech., Vol. COM - 17, No. 3, June 69.

/16/ 'Error performance of quadrature pilot - tone phase - shift keying', Bussgang, J.; Leiter, M.; IEEE Trans. Comm. Tech., Vol. COM - 16, No. 4, August 68, pp. 526 - 529.

/17/ 'The Fadeogram : An ionogram - like display of the time - varying frequency response of HF SSB radio channels', Filter, J.H.J.; Arazi, B.; Thomson, R.G.W.; IEEE Trans. Comm. Tech., Vol. COM - 26, No.6, June 78.

/18/ 'The Fadeogram - Applied to analysing HF data modem performance', Thomson, R.G.W.; Filter, J.H.J.; National Electrical Engineering Research Institute of the South African Council for Scientific and Industrial Research, Johannesburg.

/19/ 'Effective Application of Forward - Acting Error - Control Coding to Multichannel HF data modems', Pierce, A.W.; Barrow, B.B.; Goldberg, B.; Tucker, J.; IEEE Trans. Comm. Tech., Vol. COM - 18, No. 4, August 1970.

/20/ 'Error patterns measured on transequatorial HF communication links', Brayer, K., IEEE Trans. Comm. Tech., Vol. COM - 16, No. 2, April 1968.

/21/ 'Improved signal design for fading channels by coding', Chase, D., CNR Inc., Newton, Mass.

/22/ 'Application of forward - error correction to a Rayleigh fading communication channel', Treciokas, R., Proc. IEE, Vol. 125, No. 3, March 1978, pp.173 - 178.

/23/ 'The Improvement of Digital HF Communication through coding', Brayer, K., IEEE Trans. Comm. Tech., Vol. COM - 16, No.6, December 1968.

/24/ 'HF FSK error rate measurements', Ames, J.W., IEEE Trans. Comm. Tech.,Vol. COM - 17, August 69, pp. 438 - 442.

/25/ 'On a class of Error - Correcting Binary Group Codes', Bose, R.C.; Ray - Chaudhuri, D.K.; IEEE Trans. on Information and Control, Vol. IC - 3, 1960, pp. 279 - 290.

/26/ 'Computer Implementation of decoders for several BCH codes', Michelson, A.M., Polytech. Inst. of Brooklyn International Symposium on Computer Processing in Communications, April 69, pp. 401 - 413.

/27/ 'A 500 - stage CCD Transversal Filter for Spectral Analysis', Brodersen, R.W.; Hewes, C.; Buss, D.D.; IEEE Trans. Electronic Devices, Vol. ED - 23, Feb. 1976, pp. 143 - 152.

/28/ 'Microprocessor Implementation of Tactical Modems for Data Transmission over VHF radio', Davis, B.H.; Davis, B.A.; Radio and Electronic Engineer, Vol.49, No. 4, April 79, pp. 204 - 210.

/29/ 'Communications applications of CCD Transversal Filters', Buss, D.D.; Broderson, R.W.; Hewes, C.; Tasch, A.F.; IEEE Nat. Telecomm. Conf. Rec., Vol. 1, Dec.1975, pp. 1.1 - 1.5.

/30/ 'Comparison between the CCD CZT and the digital FFT', Buss, D.D.; Veenkant, R.L.; Broderson, R.W.; Hewes, C.; Proc. 1975 Naval Electronics Lab. Center Int. Conf. on the Application of CCD's, October 1975, pp. 267 - 281.

/31/ 'Encoding and Error Correction Procedures for the Bose - Chaudhuri - Hocquenghem Codes', Peterson, W.W., IRE Trans. on Information Theory, September 1960, pp. 459 - 470.

/32/ 'Cyclic Decoding Procedures for the Bose - Chaudhuri - Hocquenghem Codes', Chien, R.T., IEEE Trans. on Information Theory, September 1964.

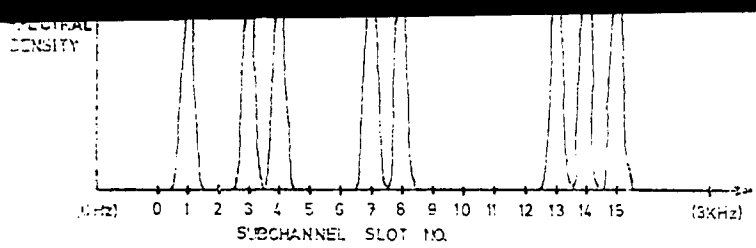


Figure 1. One possible distribution of subchannels within the voice channel

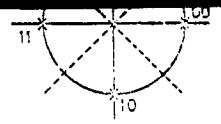
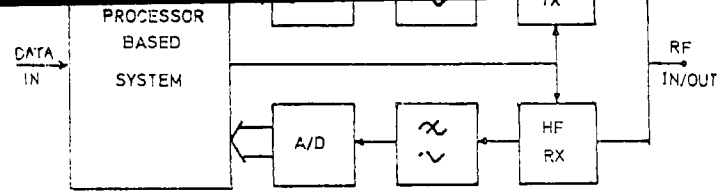
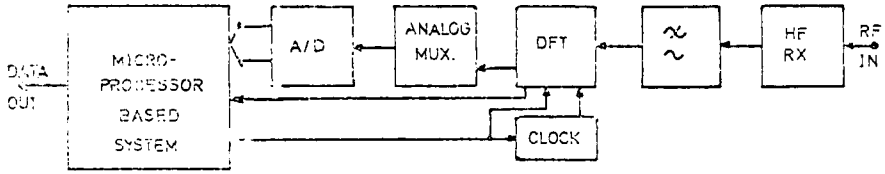


Figure 2. Phase encoding



(a)



(b)

Figure 3. Transmitter (a) and Receiver (b) systems

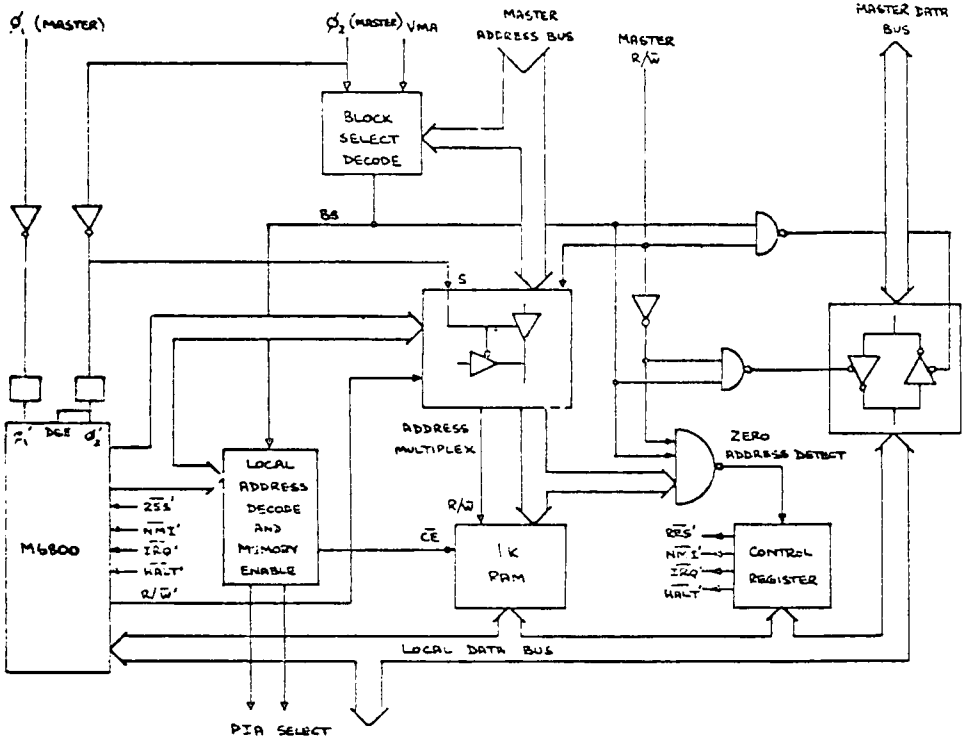


Figure 4. Slave processor system

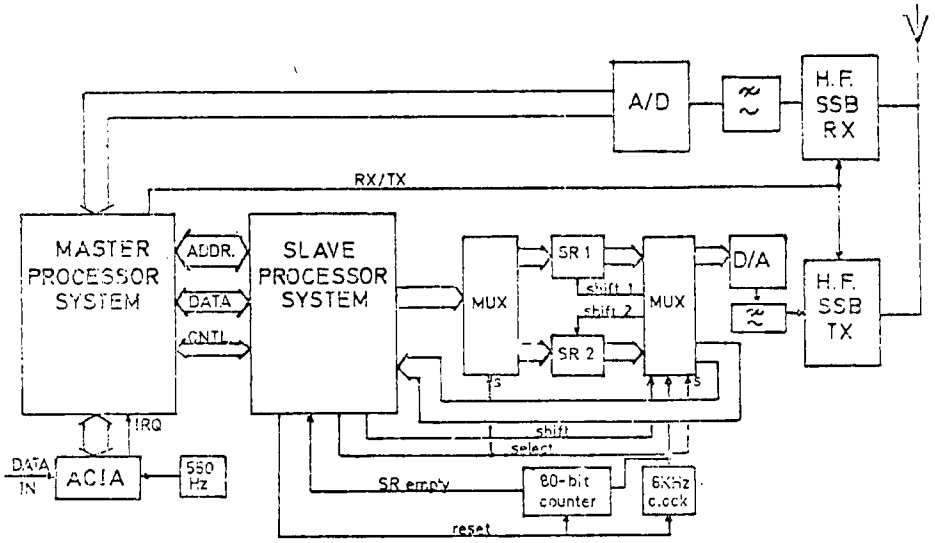


Figure 5. Transmitter system

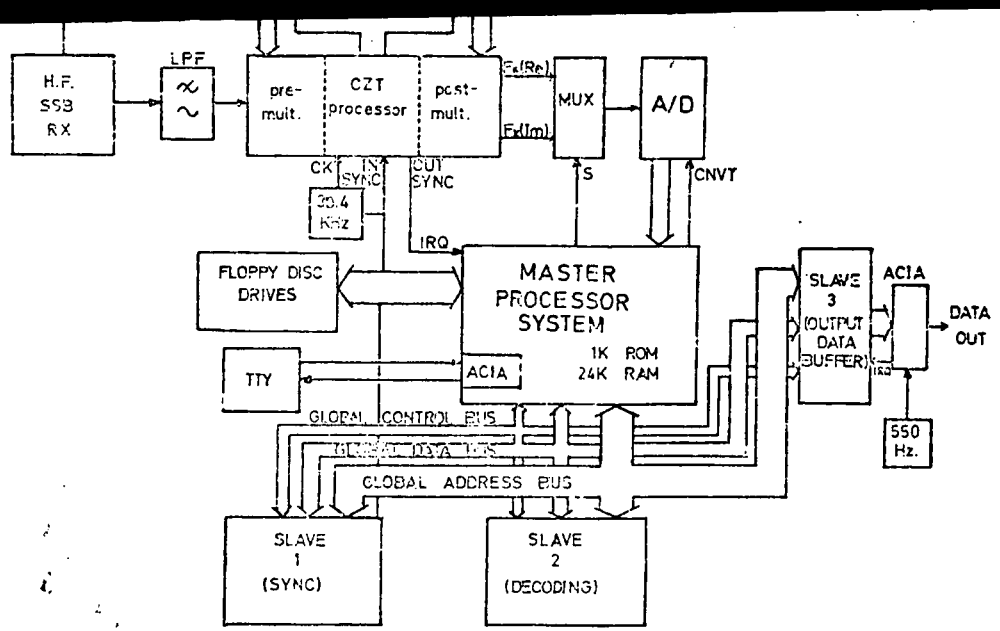


Figure 6. Receiver System

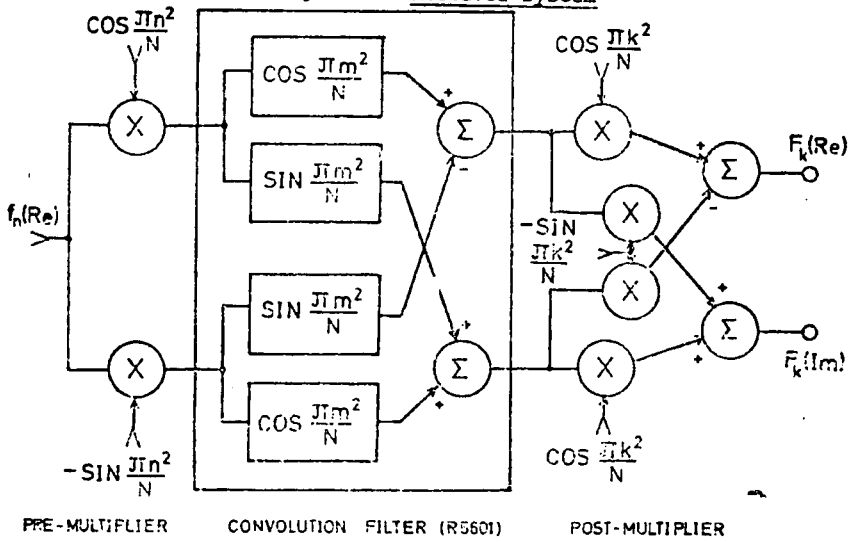


Figure 7. Implementation of Chirp-Z transform

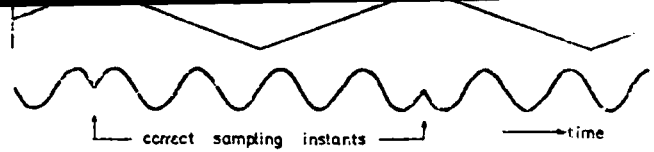


Figure 8. Variation of phasor magnitude over phase reversal sequence

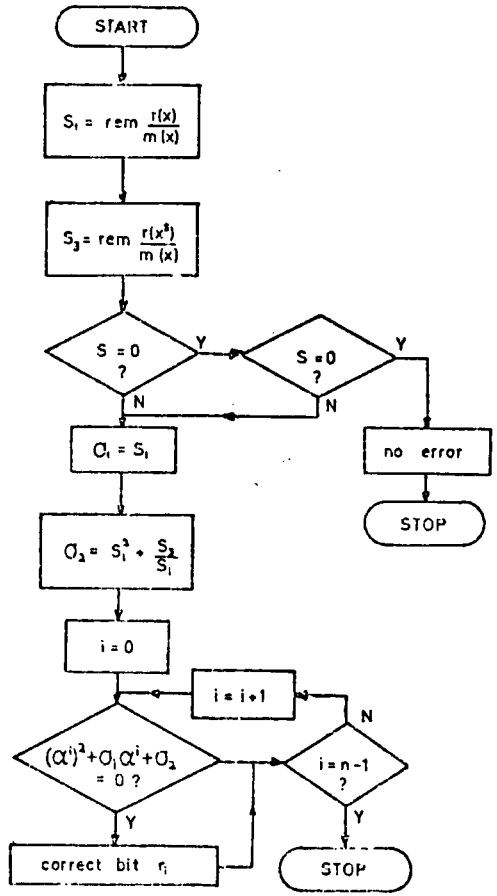


Figure 9. Implementation of decoding algorithm

Microprocessor Controlled Trackside Recorder

D.R. Isaac, C.T. Spracklen,
J.R. Manning

Abstract

This paper describes a new trackside train speed recorder designed at Durham University Applied Physics Department for the Chief Civil Engineer, British Rail Southern Region. The new recorder is controlled by a microprocessor unit and is capable of recording train speed, direction, wheel count, and time of day in an electrical memory contained within the system. The recorder is later transported to a microcomputer base station where the data is offloaded and displayed on a line printer. The new unit is entirely electronic in operation and is powered by rechargeable dry cells. The system is lighter, smaller, and more flexible than any previous designs.

History

The interest in the speeds which trains achieve extends beyond railway staff to some sections of the general public. Usually the interest is in the performance of particular trains over various routes and a record of speeds achieved can easily be obtained on board the train. Speed records in this form are of considerable value to railway operators. To be of use to permanent way engineers it is often desirable to obtain the speed of all trains passing over a particular piece of track. A man stood at the trackside using a stopwatch could obtain this information but such a practice is seldom justified.

The use of automatic unattended recording systems would be ideal for permanent way needs. The limited commercial scope for such equipment has tended to restrict development to the adaption to train speed recording of standard commercial data recorders. The few recorders that have been developed tend to suffer from the trackside environment leading to frequent mechanical failures.

Specifications

A small, battery operated unit was required which could be placed at the track side to record data over a period of several days. The following parameters were to be recorded for each train passing during that period: time of day, train speed, wheel count and direction of travel. The recorder unit would then be transported to a base computer for offloading the data accumulated during the recording period and for recharging the dry cells. The unit would then be returned to the track side for further recording.

The unit was required to measure train speeds in the range 1 to 150 mph, and to record the speed rounded down to the nearest 1 mph. Trains travelling at speeds not exceeding 1 mph were to be regarded as stationary and not recorded. The time of day was to be recorded at the instant of arrival of a train, to the nearest minute at least, and the system was to be capable of recording up to 200 axles per train. The overall storage capacity of the system memory was to be sufficient to hold data relating to 1000 trains (based on an estimate of a maximum of 250 trains per day passing over a period of 4 days).

To operate the existing recorder, two switches are mounted on the rail head, spaced 66 feet apart (figure 1). The front wheel of an approaching train causes these switches to close in turn. The train speed is recorded by measuring the time interval between switch closures. The new system was required to operate correctly using the existing track switches but should be adaptable to other spacings.

A single base station may serve a number of trackside units, the overall cost being considerably lower than any previous system.

Design and Construction

Recent advances in silicon integrated circuit technology have allowed complex logic operations to be implemented using very small amounts of hardware. Instructions specifying the operations to be performed are stored sequentially in a "read only" memory. These instructions are read from the memory by a microprocessor and are obeyed or "executed" accordingly. Changes in the operation of the system may be effected by simply modifying the instructions (the "software") in the memory. The new trackside recorder is based on such a system; a block diagram is shown in figure 2.

The system is controlled by the microprocessor unit (a Motorola M6802), whose program is contained in a 2 kbyte programmable read only memory. The Peripheral Interface Adapter (PIA) is used to interface the microprocessor to the track switches, the control pushbuttons, and the base station. A crystal controlled clock IC is used to maintain the time of day. Train data is stored in four CMOS read/write memory ICs, which are powered by a backup battery supply in the event of main battery failure. A Liquid Crystal Display, with its associated interface IC, is included to display the current time, or the speed of the last train recorded.

The system software continuously monitors the track switches for a closure. When a closure is detected, the train direction is known, and the second switch is monitored for closure. The time between closures is measured to an accuracy of 1ms, and the train speed is computed by dividing a fixed number (which is proportional to the switch spacing) by the measured time. The number of axles is then found by counting the number of successive closures of the second switch (a 15ms "debounce" time

is allowed for each closure). It is assumed that the train has passed if no switch closures are detected within a time which is inversely proportional to the train speed. After the last axle has passed, the data is stored in memory in the format to be described, and a delay of exactly 15s is allowed before commencing the search for a new train. If a switch closure is detected within this time, the axle count is regarded as erroneous and is stored as a count of '0'. The search for a new train will not begin until both switches have returned to the rest position.

Data for each train is stored in four or five bytes (8 bits) of read/write memory. The fifth byte is used only if there has been a change in hours since the previous train was recorded. Otherwise the hours are not recorded (to conserve memory). Minutes and seconds are recorded using 7 bits each, direction is recorded as a single bit, speed and axle count occupy one byte each. Up to 254 axles per train are permitted. The maximum memory capacity is 8 kbytes, which allows nearly 1000 trains to be recorded. If a smaller memory capacity is sufficient, the number of memory ICs may be reduced accordingly to save cost.

Eight data lines and two control lines are used to interface the recorder unit to the base station for data offloading. A logic '0' on one of these lines (from the base station) indicates that the offloading procedure is to begin. Successive bytes of data are then passed from the recorder to the base station on command until the end of the data table is encountered. The batteries may then be changed, and the system time adjusted by depressing one of two buttons located on the inside of the recorder housing for "slow" or "fast" time advance (the time is monitored on the LCD mounted on the front panel).

The major elements of the recorder unit are mounted on a dual-sided printed circuit board, which also contains the backup battery supply. Several pushbutton controls are available inside the housing: Reset, to restart the recording operation; Test buttons, for simulation of the trackside switches; Slow and fast clock advance, for presetting system clock. A single control marked "time" is available outside the unit. Ordinarily, the speed of the last passing train is presented on the liquid crystal display; if the "time" button is depressed, the current system time is displayed until the button is released.

Base station

The system base station was required to offload, print, and if necessary, recall the data from the trackside unit. It was centred around an AIM-65, a small, inexpensive, 8-bit microcomputer system, with an integral printer. The trackside recorder base station program prompts the user to connect the recorder unit to the computer, then prints out the results in the format shown in figure 3. This format may be altered by simple software changes. A change of batteries will normally be made after the data has been offloaded. It is, however, possible to change the batteries before, even while the recorder is in the

field, as the battery backup will protect any existing data.

Field Trials

Laboratory tests have shown the accuracy of the new recorder to be 0.002% at 1 mph and 0.33% at 150 mph with a 66 ft. switch spacing. A comparison with an existing recorder over the full range of speeds showed the new recorder to be considerably more accurate, especially at higher speeds. accurate then the GMT at higher speeds. Tests on a stretch of (third rail) electrified line have shown the system to be immune to the transient effects often produced in a high-intensity electric field environment. Field tests have been carried out using a variety of switch mechanisms. The Silec oil-damped treadles, normally used with the existing recorders, gave good speed results but spurious axle counts due to the long time required for the treadles to return to rest after closure. Removal of the oil damping cures this problem but considerably shortens the life of the treadles. A magnetic proximity-detector type mechanism again gave good speed measurements but generally produced an axle count greater than the true count due to excessive contact bouncing. A pair of pressure-sensitive microswitches advantages of small size and low cost. Work is now in progress to fully develop this type of track switch.

Future Development

Automatic train identification should be possible if a measurement of axle spacing is made in addition to the axle count. The measured data can be correlated with stored data relating to a number of train types. The train type may then be identified with a high degree of certainty. This information would then be printed at the base station, together with the speed, direction and time.

A multiple switch facility would allow a number of switch pairs to be connected to a single recorder unit in order to monitor several tracks simultaneously. This might be especially useful for monitoring train movements over a number of routes at complex rail junctions. A maximum of four switch pairs would be tolerable without major hardware modifications to the current system. However, considerable software modifications would be necessary for this option.

The power consumption of the current trackside recorder system is approximately 1W; most of this is due to the M6802 microprocessor. A CMOS plug-in replacement for this device has been announced, which should reduce power consumption, and hence battery drain, by approximately 66%. It will shortly be possible to replace the other major components, namely the PIA and the EPROM, with their CMOS equivalents which should then allow the unit to operate continuously for several months without the need for battery recharging.

Conclusion

A trackside train speed recorder based on microprocessor technology has been described which presents many advantages over previous designs. A cost effective system is achieved, together with an overall performance improvement. The new recorder unit can be used with a variety of switch mechanisms and switch spacings, and may be operated from a set of dry batteries for several days. Very favourable results have been obtained from tests in the laboratory and in the field. It is anticipated that the falling cost of microelectronics technology will lead to further improvements and cost reductions in the near future.